EFFICIENT TECHNIQUES FOR SECURE MULTIPARTY COMPUTATION ON MOBILE DEVICES

A Thesis
Presented to
The Academic Faculty

by

Henry L. Carter

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science

Georgia Institute of Technology
December 2015

EFFICIENT TECHNIQUES FOR SECURE MULTIPARTY COMPUTATION ON MOBILE DEVICES

Approved by:

Professor Patrick Traynor, Advisor
School of Computer Science
Georgia Institute of Technology

Professor Chris Peikert
School of Computer Science
Georgia Institute of Technology

Professor Mustaque Ahamad
School of Computer Science
Georgia Institute of Technology

Professor Kevin Butler
Department of Computer and
Information Science and Engineering
University of Florida

Professor Sasha Boldyreva
School of Computer Science
Georgia Institute of Technology

Date Approved: 21 October 2015

To my parents, Fray and Susan,

who inspired me to excel and encouraged me to persevere.

ACKNOWLEDGEMENTS

Before all else, I thank my parents for the years and effort they dedicated towards raising and educating me to be the man I am. Without the seeds of curiosity, self-improvement, and excellence that they planted, I would never have even begun the long road towards the achievement of a doctorate. And because they know me better than anyone else, their words of encouragement kept me moving towards the degree even when I was most discouraged.

I next want to thank my advisor for these years of mentorship and provision. He has opened every opportunity possible along the way, allowing me to grow both intellectually and professionally. His mentorship and advice has led me through grueling lessons and set me afloat as my own, self-sufficient academic. I am also grateful to my committee for the time and effort they have put towards evaluating and directing my research. Their guidance and support have honed this final accomplishment as a student into work that reflects my progress over the years.

Finally, I thank my friends who provided the everyday companionship and support to my morale in the trenches of grad school. Without collaborators along side to stick out the long nights working in the lab, this dissertation would never have made it past a sketch on a whiteboard. Without fellow grad students to share in my defeats and my victories, the pursuit of academic excellence would have been a lonely and hollow trudge. Without close friends to keep me grounded and enjoying life in the world outside of computing, I would have burned up my energy and dropped out of the marathon of graduate school long before the finish line.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Smartphones are rapidly becoming a widespread computation platform, with many users relying on their mobile devices as their primary computing device. This popularity has brought about a plethora of mobile applications and services which are designed to efficiently make these limited devices a viable source of entertainment and productivity. This is commonly accomplished by moving the critical application computation to a Cloud or application server managed by the application developer. Unfortunately, the significant number of breaches experienced by mobile application infrastructure and the accompanying loss of private user data indicates the need for stronger security and privacy guarantees before this model of computation can become ubiquitous.

The cryptographic community has developed the field of secure multiparty computation (SMC) to allow applications to perform computation over encrypted data. Such a protocol would allow mobile users to keep their private information encrypted while still enjoying the convenience of their Cloud based applications. However, while SMC protocols have seen significant advances in efficiency on desktop and server class machines, they currently require more computation power and memory than is available on commodity smartphones. Furthermore, even as smartphone computational power increases, the mobile-specific limitations of network bandwidth and power usage will always stand as barriers to efficiently executing SMC protocols.

This dissertation develops techniques for outsourcing the costly operations in garbled circuit SMC protocols to an untrusted Cloud to allow resource-constrained devices to use this cryptographic primitive. By providing the mobile device with a third party Cloud provider, we show that it is possible for a mobile device to execute a garbled circuit with an application server at approximately the same efficiency as the same computation run between two server class machines. We first show two protocols for outsourcing the garbled circuit evaluation and generation. We develop a novel outsourced oblivious transfer (OOT)

protocol to make this type of outsourcing possible. Second, we develop a black box technique for outsourcing *any* two-party SMC protocol, and show that the overhead incurred by outsourcing is minimal. Finally, we develop a protocol for outsourcing SMC that provides both input privacy *and* circuit privacy, preventing the assisting Cloud from learning anything about the computation besides the fact that it took place. Through the protocols and the empirical evaluations in this dissertation, we show that executing SMC protocols on mobile devices can be done with comparable efficiency to the desktop platform, and provide techniques to allow for such computation using the latest developments in secure computation.

CHAPTER I

INTRODUCTION

As modern computing trends move towards more mobile and cloud computing, data privacy is more important than ever before. With smartphones constituting over 57% of the U.S. market share [44] and with over 1.4 billion in use worldwide [108], users are running applications that routinely send private information to application servers. Location-based services, social networks, and banking apps are a few examples of applications that process private information. Outside of mobile computing, cloud storage and processing have become convenient ways for individuals and corporations to outsource their computing needs to services such as Amazon EC2 and Microsoft Azure. However, this convenience comes at the cost of losing privacy to the corporation that is hosting these cloud services. A growing record of cloud data breaches clearly demonstrates that these corporations are struggling to maintain a requisite level of privacy for user data [154, 155, 144, 94]. Finally, as news continues to break regarding government surveillance on digital communication, it is clear that companies like Google and Facebook, who manage the services that millions use to communicate online, cannot provide the necessary security to prevent government mandated data seizure. Given that it is highly likely that outsourcing computation will only grow in popularity, the issue of maintaining data privacy while preserving the convenience of cloud computing is a critical problem.

One possible solution to this problem is secure multiparty computation (SMC). First demonstrated possible with the Yao garbled circuit in 1986 [160], SMC protocols allow two or more mutually distrustful parties to compute a shared result while keeping their respective inputs hidden from the other participating parties. A number of different techniques have been used to achieve these impressive security guarantees, including homomorphic encryption [63], secret-sharing schemes [21], and Oblivious RAM [71]. In recent years, new research has moved the Yao garbled circuit construction from a theoretical novelty to the

most efficient technique for 2-party SMC [151, 109, 79]. However, while the techniques for executing garbled circuit protocols have come a long way, there are still many efficiency roadblocks that are preventing these techniques from being applied in practical applications. Specifically, the process of garbling, sending, and evaluating a boolean circuit of any useful size still requires significant computational power, memory, and bandwidth [32]. In addition, one of the primitives used to build these protocols, Oblivious Transfer (OT), is a notably expensive operation because of the asymmetric cryptographic primitives it uses. To achieve security against malicious adversaries, these costs are magnified through the use of cut-and-choose techniques. Given these restrictions, it is impractical for cloud services and nearly impossible for mobile applications to implement garbled circuit protocols.

The goal of this dissertation is to improve the efficiency of garbled circuit SMC protocols enough to begin seeing practical implementation on the mobile platform. Rather than constructing new primitives for garbled circuit protocols, we develop outsourcing techniques that allow existing protocols to be executed more efficiently and that can be applied to new garbled circuit protocols as they are developed. It is critical for these techniques to improve efficiency with respect to not only computation time, but all system resources, including memory and bandwidth usage. Additionally, it is critical that these improvements account for real-world security requirements and assumptions to ensure that they can be directly and simply implemented in practice. The first part of our work demonstrates how existing techniques for evaluating garbled circuits can be ported to mobile devices with the help of an outsourcing cloud. This allows mobile devices to participate in garbled circuit computation at approximately the same efficiency as desktop-class machines, with minimal computation, memory, and bandwidth cost. We then show that under specific adversarial assumptions, we can increase the security of outsourcing SMC to prevent the Cloud from learning anything about the circuit being evaluated, in addition to maintaining input and output privacy.

1.1 Thesis Statement

The goal of this work is to bring SMC into the realm of practical use on the mobile platform. Given the rapid development of new SMC techniques, we believe the focus of research

should be on applying the existing garbled circuit protocols in ways that meet the security needs of real-world applications, and will continue to provide efficiency improvements as the underlying garbled circuit protocols are improved and replaced. Given these goals and building on our preliminary work, we present the following thesis statement:

Outsourcing costly cryptographic operations for SMC to untrusted Cloud infrastructure can provide equivalent security to non-outsourced protocols and significantly improves mobile protocol performance, allowing resource-constrained devices such as smartphones to run real-world applications using these privacy-preserving primitives.

1.2 Contributions

Prior to this work, the performance of SMC protocols on the mobile platform had not been characterized, and no efficient techniques for mobile SMC existed. This dissertation provides three major contributions:

- **Characterization of mobile SMC:** We first show that existing SMC protocols are too computationally expensive for the resource-constrained mobile platform. Furthermore, our work demonstrates that the high bandwidth costs of these protocols is likely to make them inefficient for mobile devices well into the future.

- **Develop SMC outsourcing techniques:** Our work constructs a set of protocols that can outsource the most costly operations associated with SMC securely from a mobile device. We demonstrate that these protocols are secure against malicious adversaries, and show how new SMC protocols can be outsourced in the same security setting.

- **Develop Private Function Evaluation for outsourcing:** We examine specific adversary constraints that can allow for SMC outsourcing without revealing the circuit being evaluated to the Cloud. This construction allows for efficient and maximally secure outsourcing in a variety of real-world application scenarios.

1.3 Organization

This dissertation is organized as follows: Chapter 2 describes the related literature; Chapter 3 presents our protocol for outsourcing garbled circuit evaluation and a characterization of garbled circuit evaluation on the mobile platform; Chapter 4 presents our protocol for outsourcing garbled circuit generation; Chapter 5 presents our protocol for outsourcing any two-party SMC protocol as a black box; Chapter 6 presents our set of protocols for outsourcing SMC with both function and input privacy; and Chapter 7 provides concluding remarks and discusses future research opportunities.

CHAPTER II

BACKGROUND

2.1 Secure Multiparty Computation

Since the first protocols were developed by Andrew Yao [159], research in secure multiparty computation has developed into a broad and diverse field of research. Encompassing constructions such as oblivious transfer [143] and private information retrieval [42, 106], SMC generally describes any technique for evaluating some functionality in a privacy-preserving way. The security provided by these schemes is most commonly demonstrated using a real/ideal world paradigm, which proves that in a real world execution of the SMC protocol, participants output a computationally indistinguishable set of values from an ideal world where the function is evaluated by a trusted third party, who then distributes some output to all participants [66]. While many variations on this paradigm are used to prove the security of different constructions, this basic concept intuitively demonstrates that the SMC protocol provides equivalent security to the best achievable solution of a fully trusted third party. We provide the outsourced formulation of this security model in Chapter 3.

This work develops protocols for privacy-preserving computation of arbitrary functions. The existing protocols to achieve such generic SMC constructions can be divided into three categories of techniques: Secret-Sharing, Homomorphic Encryption, and Garbled Circuits.

2.1.1 Secret-Sharing

Secret-sharing SMC techniques encompass a variety of protocols that allow for private data to be split into encrypted shares and divided between the participants. These shares are then processed in a privacy-preserving interactive protocol and combined at the end to recover the resulting output. These protocols commonly require some random data (e.g., multiplication triples) to be encrypted in a pre-processing phase, which is then combined with the secret shares during the online computation to allow for non-linear operations such as multiplication in the arithmetic setting or bitwise AND in the boolean setting. The

GMW construction [67] developed one of the earliest secret-sharing techniques, which allowed for computation over boolean circuits, and required an oblivious transfer protocol to be executed for computing AND gates. While the extensive use of OT in this protocol has commonly been seen as prohibitively costly in terms of communication complexity, recent developments have shown that the GMW protocol can be quite efficient for evaluating low-depth boolean circuits [129, 132, 149, 40, 6, 140, 107]. In addition, many arithmetic secret-sharing protocols and optimizations have been developed using a wide range of underlying secret sharing protocols [12, 7, 141, 52, 29, 65]. These optimizations have been augmented with further research into constructing optimal arithmetic circuit representations for common functions [36, 37]. However, secret-sharing protocols are generally optimal for large numbers of participants, and tend to be less practical in the two-party setting.

2.1.2 Homomorphic Encryption

Homomorphic encryption includes a range of encryption schemes that satisfy one or more homomorphic operations over the ciphertext. Specifically, given an encryption scheme $(Gen(\cdot), Enc_{key}(\cdot), Dec_{key}(\cdot))$, an operation $\diamond$ over the plaintexts, and an operation $\star$ over the ciphertexts, it holds for any two messages m_0, m_1 that:

$$Dec_k(Enc_k(m_0) \star Enc_k(m_1)) = m_0 \diamond m_1$$

For example, an additive homomorphic scheme sets the operation $\diamond$ to addition, and allows for the addition of plaintext messages while encrypted. Schemes that allow for one homomorphic operation have been known since the original RSA construction [146]. Termed partially homomorphic encryption, these techniques have yielded several special-purpose protocols for computing specific functions [135, 32, 28, 75, 122]. However, without supporting a universal set of homomorphic operations, these schemes do not allow for generic computation over encrypted data.

To evaluate arbitrary functions over encrypted data, the concept of a fully-homomorphic encryption (FHE) scheme was proposed by Rivest et al. [145]. To satisfy the current definition for FHE, an encryption scheme must provide the following three properties. First, it allows for a universal set of homomorphic operations, such as addition and multiplication.

Second, it allows for evaluation of arbitrary-depth circuits. Finally, the ciphertext size is not dependent on the size of the function that produced it. The first scheme to achieve these properties was developed by Gentry [63, 62] by using the concept of a bootstrappable encryption scheme. This concept is built on the use of somewhat homomorphic encryption (SHE), which is an encryption scheme that allows for a universal set of homomorphic operations, but can only evaluate circuits of a limited depth. As long as the depth limit of the SHE scheme is greater than the depth of the decryption circuit for that scheme, the ciphertext can be decrypted homomorphically, producing a fresh ciphertext that can accept more homomorphic operations. This process of homomorphic decryption is called bootstrapping. Gentry instantiated his SHE scheme using a lattice construction, which has been a common primitive across many more recent homomorphic encryption constructions [25, 45, 2, 23]. More recently, leveled-FHE schemes have been developed that eliminate the need for bootstrapping [24, 64]. While these schemes can evaluate circuits of arbitrary depth, that depth d must be specified when the key is chosen, after which the depth of the evaluated circuits must be less than or equal to d. Beyond standard computation over encrypted data, FHE has been used as an underlying primitive in other constructions, such as attribute-based encryption [22] and functional encryption [60], as well as applications such as reusable garbled circuits [69], verifiable computing [61], and homomorphic signatures [70]. While these cryptosystems have developed significantly in only a few years of research, they are currently too inefficient to be used practically, even on server-class machines.

To apply homomorphic encryption schemes for more practical SMC protocols, several techniques for combining SHE with secret-sharing techniques have produced multiparty protocols that are significantly more efficient than FHE [19, 47, 46, 50, 95]. By using SHE during a preprocessing phase to produce a set of multiplication triples, these protocols allow for a highly efficient online phase that is optimized for computation between a large number of participants. Unfortunately, the costly preprocessing phase and the loss of efficiency in the two-party case makes these protocol less practical for our mobile setting.

2.1.3 Garbled Circuits

The garbled circuit SMC protocol that we apply in this dissertation was developed by Andrew Yao [160], and provided the first protocol for evaluating arbitrary functions in a privacy-preserving manner. Using only a symmetric encryption scheme and an oblivious transfer protocol, Yao's protocol allows functions represented as boolean circuits to be obliviously evaluated in a constant number of communication rounds. The protocol requires at least two participants. The first is the generator, who is responsible for obscuring the bit values and gate functionality for the chosen function to be evaluated. The evaluator is then given a set of garbled input values and the garbled circuit, and is responsible for obliviously evaluating the circuit. The protocol proceeds as follows:

1. **Garbling:** For every wire in the circuit, the generator creates two random strings w_i^0, w_i^1, which are the garbled wire labels for the bit values 0 and 1 on the i^{th} wire. Next, he garbles each gate by encrypting the entries in a truth table that corresponds to the gate's functionality. For simplicity, we only consider two-input gates, but the same operations can be used to garble a gate with any number of inputs or outputs. For a gate executing the arbitrary boolean operation $\star$ which takes input wires i and j and outputs on wire k, he encrypts each entry as:

$$Enc_{w_i^{b_i}||w_j^{b_j}}(w_k^{b_i \star b_j})$$

   where b_i and b_j are the logical bit values for wires i and j. After permuting the entries in each garbled truth table, the generator sends all of the garbled gates and the input wire values that correspond to his secret input to the evaluator.

2. **Oblivious Transfer:** For each of the evaluator's secret input bits b_i, the generator and evaluator execute a 1-out-of-2 oblivious transfer. This protocol allows the evaluator to receive the garbled wire label $w_i^{b_i}$ without the generator learning the secret value b_i. Moreover, the evaluator learns nothing about the wire label $w_i^{1-b_i}$.

3. **Evaluation:** Given the garbled input values for both parties and the garbled gates, the evaluator can obliviously evaluate the circuit. For each gate, she possesses the

correct wire labels to decrypt exactly one entry in the garbled truth table, which allows her to decrypt subsequent entries in subsequent gates.

4. **Output:** Once the evaluator possesses wire labels for each of the output wires, she can deliver these garbled wire values back to the generator, who can recover the output using his original mappings between garbled wire labels and logical bit values. The generator can then optionally deliver this output to the evaluator.

The privacy guarantee of Yao's protocol is based on the fact that the generator never observes any of the intermediate wire labels used during evaluation, while the evaluator never learns the mappings between garbled wire labels and logical wire values [114]. However, this basic protocol only ensures security against semi-honest adversaries who are guaranteed not to deviate from the prescribed protocol.

Beginning with Fairplay [119], several garbled circuit-based SMC implementations and applications have been developed in the semi-honest adversarial model [105, 86, 82, 90, 110, 113, 118, 134, 16]. However, a malicious party using corrupted inputs or circuits can learn private information about the other party's inputs in these constructions [98]. To resolve these issues, new protocols have been developed to achieve security in the malicious model, using cut-and-choose constructions [119, 123, 112, 158, 150, 115, 109, 57, 79], input commitments [150], and other various techniques [123, 124, 139, 99, 83, 85, 103]. To provide the ability to exchange security for efficiency, the covert security model was developed to provide security guarantees against an adversary who will only cheat if it is possible to do so without discovery [9, 48, 72]. The latest garbled circuit protocols focus on amortizing the cost of the cut-and-choose by batching several instances of the same computation [116, 84]. To improve the performance of these schemes in both the malicious and semi-honest adversarial models, a number of optimization techniques have also been developed to reduce the cost of generating and evaluating circuits [101, 41, 128, 109, 79, 104, 15].

As general SMC protocols for each of these underlying techniques become more efficient, a new set of protocols has started to develop which switches between techniques based upon which technique is optimal for the current function being evaluated [77, 54]. However,

these schemes require a significant amount of manual inspection to ensure that the best SMC technique is being used for each step of the function being evaluated. One early effort towards automated protocol compilation was made by Kerschbaum et al. [97]. While these combination techniqes allow for significant improvements in the performance of the protocol, the lack of efficient automated techniques for compiling functions into a hybrid SMC protocol makes these combination protocols difficult to use in practice.

2.2 Oblivious Transfer

A critical underlying protocol used in many SMC constructions is the oblivious transfer (OT) protocol [143]. While generalized k-out-of-n oblivious transfer schemes exist, we focus our attention on the basic 1-out-of-2 OT that is required to securely execute the garbled circuit protocol. A 1-out-of-2 OT protocol takes place between two participants, the sender and the chooser. The sender inputs two values w^0, w^1, while the chooser inputs a selection bit b. At the end of the protocol, the sender receives nothing while the chooser receives w^b. The privacy guarantee that must be ensured by this primitive applies to both parties. Specifically, the OT must guarantee that the sender learns nothing about the selection bit b, while the chooser learns nothing about the value w^{1-b} that she did not select.

Since the primitive was originally conceived, several OT schemes built on a wide range of underlying assumptions have been developed [17, 130, 115, 59], including a round-optimal construction using lattices by Peikert et al. [137]. However, the most significant OT developments to allow for improved garbled circuit protocols are OT extensions [13]. The Ishai et al. OT extension [87] allows *any* base OT scheme to be extended to reduce k^c oblivious transfers to k oblivious transfers for any given constant c. This is done by exchanging random seeds during the base OT, then using those seeds to add and remove some type of encryption from the values provided by the sender later in the protocol. Using OT extensions, delivering the garbled values corresponding to potentially large inputs became much less costly in terms of cryptographic operations and network overhead. Several more recent improvements to the Ishai et al. extension have further reduced the cost by as much as 2-3 times [100, 6]. In some settings, the online computation time of oblivious transfers can be

further reduced by precomputing the OT [11]. Even with these drastic improvements in efficiency, oblivious transfers still tend to be a costly step in evaluating garbled circuits.

2.3 Private Function Evaluation

To provide function privacy along with input privacy, a specialized set of SMC protocols have been developed for private function evaluation (PFE). The first constructions to provide both of these privacy guarantees used existing SMC constructions to evaluate a universal circuit that accepts the desired function as an input parameter [157, 1, 102, 148]. Because the specific function being evaluated is treated as an input to the universal circuit, the input privacy of the SMC protocol protects both the inputs and the function. However, the use of a universal circuit incurs significant overhead in the circuit size beyond the size of the function being computed. To provide a more efficient solution, many specialized PFE protocols were developed to provide function privacy for certain classes of functions [88, 27, 10, 136]. The first general PFE protocol to provide linear asymptotic efficiency with respect to circuit size was developed by Katz and Malka [93]. Using techniques reminiscent of the LEGO SMC construction [133], they show how a generating party can generate a set of garbled gates that are then assembled and evaluated by the party possessing the function to be computed. In more recent work, Mohassel et al. [126] demonstrated how this idea can be generalized for various underlying SMC primitives, and later extended this concept for secret sharing protocols to be secure against active adversaries [125]. However, these protocols assume a significant amount of computational power for all parties participating in the protocol.

2.4 Mobile Privacy

With smartphone applications retrieving private user data at an increasing rate, SMC could potentially offer a way to maintain privacy and functionality in mobile computing. However, the efficiency challenges of SMC are compounded when considered in the resource-constrained mobile environment. Previous work has shown that smartphones can evaluate simple functions using garbled circuits in the semi-honest model [80]. However, to thoroughly examine the capability of smartphones for common mobile-specific applications, our preliminary work developed a set of protocols using partially homomorphic encryption to

perform privacy-preserving proximity tests and set intersection [32]. We then compared the performance of several general purpose garbled circuit protocols to our custom protocols to evaluate the optimal approach to privacy-preserving mobile computation.

Our evaluation demonstrated that even for these simple applications, as input sizes and circuit complexity increased, the cost of garbled circuit protocols became prohibitively expensive in terms of memory consumption and computation time. Furthermore, we showed that the high network costs of garbled circuit protocol could potentially pose practical concerns due to mobile carrier-imposed bandwidth caps or rapid depletion of battery power. While this early work seemed to indicate that custom protocols offered a viable choice for privacy-preserving smartphone computation, our custom protocols highlighted the significant computational load imposed by the complex mathematics used in most partially homomorphic encryption schemes. And while the bandwidth required to execute our protocols was less than that of the garbled circuit constructions, the large ciphertexts of our underlying public-key encryption schemes could be a significant burden on the mobile bandwidth for more complex functions. Finally, the cost of designing a custom protocol for every potential application poses a significant barrier that most application developers who are not experts in the field cannot overcome.

In recent work, Demmler et al. [53] showed how to incorporate pre-computation on hardware tokens to improve efficiency on mobile devices in the semi-honest setting. In addition to the cost of evaluating SMC protocols, Mood et al. [128] and Kreuter et al. [104] demonstrated that even with significant optimization, the task of compiling circuits on the mobile device can also be quite costly.

The final obstacle for mobile SMC constructions is increasing security to a more realistic adversary model. Our initial characterization, as well as all of the previous work examining mobile SMC, has only considered protocols in the semi-honest adversarial setting. The cut-and-choose constructions required to increase security to the covert or malicious model multiply the amount of computational power and bandwidth required by a significant factor, further reducing the ability of these resource-constrained devices to participate in privacy-preserving computation. To get over these hurdles, this dissertation examines the potential

for securely outsourcing the most costly operations.

2.5 Outsourced Computation

The idea of outsourcing computationally-expensive tasks to a more capable machine has existed in computer science for many decades. Among the earliest examples of outsourced computation is the mainframe architecture, which allowed computationally weak terminals to connect to a powerful mainframe computer over a network [26]. Although the cost of high-powered computing equipment was high, mainframes allowed users the convenience of operating their own terminal while consolidating cost into a single, centralized server. The next advances in distributed computing techniques allowed these single centralized mainframes to be spread into distributed server clusters [38]. As personal computers developed and became powerful enough to be used without mainframe support, the mainframe and terminal architecture was relegated to supercomputing clusters and disappeared from everyday use. However, new architectures for outsourced computation took advantage of these powerful end user systems. Distributed architectures such as GRID computing [56] and heterogeneous clusters [3] allowed for very large computation tasks to be outsourced across a peer-to-peer network of end user systems. Some notable ongoing projects using this architecture are SETI at home [156, 4] and the Great Internet Mersenne Prime Search [121]. More recently, the advent of mobile computing and the near-ubiquitous availability of high-speed network connectivity have brought outsourcing similar to the mainframe architecture back as cloud computing [142, 30]. Cloud computing has again made outsourcing cost-effective for applications such as on-demand infrastructure [74] and distributing data-intensive processing [55]. However, its use in providing complex application functionality to mobile devices is most relevant to this work, as it demonstrates how a weak device with network connectivity can execute a complete set of applications by sending expensive computation to a remote server [43, 152, 153]. Unfortunately, the lack of privacy guarantees for this type of outsourcing will prevent cloud-based applications from becoming ubiquitous unless stronger security can be provided [5, 147].

2.6 Server-Aided Cryptography

To further improve the speed of cryptographic protocols on devices with minimal computational resources, the idea of outsourcing cryptographic operations has been explored for many years in the field of Server-assisted cryptography [120, 14]. Naor et al. [131] develop an oblivious transfer technique that sends the chooser's private selections to a third party, termed a Proxy Oblivious Transfer. More recently, Green et al. [73] developed a technique for outsourcing the costly decryption of attribute-based encryption schemes to the cloud without revealing the contents of the ciphertext. Atallah and Frikken [8] developed a set of special-purpose protocols for securely outsourcing Linear Algebra computations to a single cloud server. Homomorphic encryption has been used to allow secure outsourcing in data mining applications [96] as well as solving common graph problems [20]. For more generic computation, López-alt et al. developed a theoretical construction for outsourcing multiparty computation using FHE [117]. While all of these applications provide significant performance gains for specific cryptographic applications, none of them provide an efficient solutions for outsourcing general secure computation.

In concurrent work to our own, Kamara et al. [91, 92] developed two protocols for securely outsourcing the computation of arbitrary functions to the cloud. This work established the definitions of security for future outsourced SMC, demonstrated a theoretical construction for black box outsourcing, and presented the first implemented outsourced SMC protocols. However, their work was intended to demonstrate the feasibility of outsourcing in general, and to provide protocols with a complexity less than that of a two-party SMC protocol. Our work attempts to provide both stronger security guarantees and minimized computation at the mobile device. In addition to this work, Jakobsen et al. [89] developed a theoretically efficient black box outsourcing technique with a novel multiplicative MAC construction. However, their construction requires the underlying SMC protocol to allow for efficient reactive computation, which does not include a large set of the most efficient SMC protocols in existence. Furthermore, their work provides only a protocol and complexity analysis, and does not provide an implementation or any insight into the practical performance of their scheme.

CHAPTER III

OUTSOURCING GARBLED CIRCUIT EVALUATION

3.1 Introduction

SMC protocols promise to make cloud computing a practically useful model for computation, even when strict data privacy is required. However, applying standard SMC constructions to the mobile platform is a non-trivial challenge. First, these protocols universally require that both parties possess symmetric computational power, and that both parties perform work that is linear with respect to the size of the function being evaluated. Second, the mobile platform presents unique constraints beyond lower computational power and memory. Many mobile carriers limit the amount of bandwidth available to devices, and limited battery life requires that the solution be power efficient as well as computationally efficient.

A potential solution to this problem is to outsource the most costly operations in the SMC protocol to a more capable server. However, this technique introduces several technical challenges. Outsourcing introduces a new party, the Cloud provider, into the computation. This third party may or may not be trusted to handle private inputs. An ideal outsourcing solution would provide security against *both* the participating parties and the outsourcing Cloud. In addition to these security requirements, it should significantly reduce the computational burden on the resource-constrained mobile device without significantly increasing the computational burden on the servers involved.

In this chapter, we develop some of the first mechanisms for the secure outsourcing of garbled circuit SMC from constrained devices to more capable infrastructure. This protocol maintains the privacy of both the mobile user's and the application server's inputs and outputs while significantly reducing the computation and network overhead required by the mobile device for garbled circuit evaluation. We develop a number of extensions to allow the mobile device to check for malicious behavior from the circuit generator or the cloud and a novel Outsourced Oblivious Transfer (OOT) for sending garbled input data to the

cloud. We then implement the new protocol on a commodity mobile device and reasonably provisioned servers and demonstrate significant performance improvements over evaluating garbled circuits directly on the mobile device.

This protocol and the accompanying evaluation make the following contributions:

- **Outsourced oblivious transfer & outsourced consistency checks:** Instead of blindly trusting the Cloud with sensitive inputs, we develop a highly efficient Outsourced Oblivious Transfer primitive that allows mobile devices to securely delegate the majority of computation associated with oblivious transfers. We also provide mechanisms to outsource consistency checks to prevent a malicious circuit generator from providing corrupt garbled values. These checks are designed in such a way that the computational load is almost exclusively on the cloud, but cannot be forged by a malicious or "lazy" cloud. We demonstrate that both of our additions are secure in the malicious model as defined by Kamara et al. [92].

- **Performance Analysis:** Extending upon the implementation by Kreuter et al. [103], we conduct an extensive performance analysis against a number of simple applications (e.g., edit distance) and cryptographic benchmarks (e.g., AES-128). Our results show that outsourcing SMC provides improvements to both execution time and bandwidth overhead. For the edit distance problem of size 128, we reduce execution time by 98.92% and bandwidth by 99.95% compared to direct execution without outsourcing on the mobile device.

- **Privacy Preserving Navigation App:** To demonstrate the practical need for our techniques, we design and implement an outsourced version of Dijkstra's shortest path algorithm as part of a Navigation mobile app. Our app provides directions for a Presidential motorcade without exposing its location, destination, or known hazards that should be avoided (but remain secret should the mobile device be compromised). *The optimized circuits generated for this app are among the largest circuits evaluated to date.* Without our outsourcing techniques, such an application is far too processor, memory and bandwidth intensive for any mobile phone.

This work was originally published at the USENIX Security Symposium [34].

3.2 Assumptions and Definitions

To construct a secure scheme for outsourcing garbled circuit evaluation, some new assumptions must be considered in addition to the standard security measures taken in a two-party secure computation. In this section, we discuss the intuition and practicality of assuming a non-colluding cloud, and we outline our extensions on standard techniques for preventing malicious behavior when evaluating garbled circuits. Finally, we conclude the section with formal definitions of security.

3.2.1 Non-collusion with the cloud

Throughout our protocol, we assume that none of the parties involved will ever collude with the cloud. This requirement is based in theoretical bounds on the efficiency of garbled circuit evaluation and represents a realistic adversarial model. The fact that theoretical limitations exist when considering collusion in secure multiparty computation has been known and studied for many years [39, 18, 111], and other schemes considering secure computation with multiple parties require similar restrictions on who and how many parties may collude while preserving security [31, 49, 51, 92, 91]. Kamara et al. [92] observe that if an outsourcing protocol is secure when both the party generating the circuit and the cloud evaluating the circuit are malicious and colluding, this implies a secure two-party scheme where one party has sub-linear work with respect to the size of the circuit, which is currently only possible with fully homomorphic encryption. However, making the assumption that the cloud will not collude with the participating parties makes outsourcing securely a possibility in practice. In reality, many cloud providers such as Amazon or Microsoft would not allow outside parties to control or affect computation within their cloud system for reasons of trust and to preserve a professional reputation. In spite of this assumption, we cannot assume the cloud will always be semi-honest. For example, our protocol requires a number of consistency checks to be performed by the cloud that ensure the participants are not behaving maliciously. Without mechanisms to force the cloud to make these checks, a "lazy" cloud provider could save resources by simply returning that all checks verified without

actually performing them. Thus, our adversarial model encompasses a non-colluding but potentially malicious cloud provider that is hosting the outsourced computation.

Kamara et al. [91] define this property formally as a non-cooperative adversary.

Definition 1. *An adversary A_i is non-cooperative with respect to adversary A_j if the messages A_i sends to A_j reveal no information about A_i's private values to A_j beyond what can be inferred from A_j's output $f_j(x)$.*

This property ensures that the view of two adversaries is not shared in any way, even if both parties are corrupted. As demonstrated by Kamara et al. [91], this allows us to prove security using a set of partial outputs for each non-cooperative adversary (we define partial outputs later in this section). In practice, this allows us to guarantee security when all of the parties in the protocol are at least semi-honest (that is, they are *not* honest and may collude with one another unless defined as non-cooperative).

3.2.2 Attacks in the malicious setting

When running garbled circuit based secure multiparty computation in the malicious model, a number of well-documented attacks exist. We address here how our system counters each.

- **Malicious circuit generation:** In the original Yao garbled circuit construction, a malicious generator can garble a circuit to evaluate a function f' that is not the function f agreed upon by both parties and could compromise the security of the evaluator's input. To counter this, we employ an extension of the cut-and-choose technique using random seeds developed by Goyal et al. [72] and implemented by Kreuter et al. [103]. Essentially, the technique uses a cut-and-choose, where the generator commits to a set of circuits that all presumably compute the same function. The parties then use a fair coin toss to select some of the circuits to be evaluated and some that will be re-generated and hashed by the cloud given the random seeds used to generate them initially. The evaluating party then inspects the circuit commitments and compares them to the hash of the regenerated circuits to verify that all the check circuits were generated properly. Recent developments have produced significantly

improved techniques for cut-and-choose in garbled circuit protocols [109, 79]. Because the technique developed by Lindell [109] maintains separate roles of circuit generator and evaluator, we believe that it is possible to incorporate this optimization into our outsourcing protocol, and leave this improvement to future work.

- **Selective failure attack:** If, when the generator is sending the evaluator's garbled inputs during the oblivious transfer, he lets the evaluator choose between a valid garbled input bit and a corrupted garbled input, the evaluator's ability to complete the circuit evaluation will reveal to the generator which input bit was used. To prevent this attack, we use the input encoding technique from Lindell and Pinkas [112], which lets the evaluator encode her input in such a way that a selective failure of the circuit reveals nothing about the actual input value. To prevent the generator from swapping garbled wire values, we use a commitment technique employed by Kreuter et al. [103].

- **Input consistency:** Since multiple circuits are evaluated to ensure that a majority of circuits are correct, it is possible for either party to input different inputs to different evaluation circuits, which could reveal information about the other party's inputs. To keep the evaluator's inputs consistent, we again use the technique from Lindell and Pinkas [112], which sends all garbled inputs for every evaluation circuit in one oblivious transfer execution. To keep the generator's inputs consistent, we use the malleable claw-free collection construction of shelat and Shen [150]. This technique is described in further detail in Section 3.2.3.

- **Output consistency:** When evaluating a two-output function, we ensure that outputs of both parties are kept private from the cloud using an extension of the technique developed by Kiraz and Schoenmakers [98]. The outputs of both parties are XORed with random strings within the garbled circuit, and the cloud uses a witness-indistinguishable zero-knowledge proof as in the implementation by Kreuter et al. [103]. This allows the cloud to choose a majority output value without learning either party's output or undetectably tampering with the output. At the same time, the witness-indistinguishable proofs prevent the evaluator and the generator from learning the

index of the majority circuit. This prevents the generator from learning anything by knowing which circuit evaluated to the majority output value.

3.2.3 Malleable claw-free collections

To prevent the generating party from providing different inputs for each evaluation circuit, we implement the malleable claw-free collections technique developed by shelat and Shen [150]. Their construction essentially allows the generating party to prove that all of the garbled input values were generated by exactly one function in a function pair, while the ability to find an element that is generated by both functions implies that the generator can find a claw.

Goldreich and Kahan define a claw-free collection as a three-tuple of algorithms [68]. The index sampling algorithm, G, takes a security parameter 1^n as input, and specifies an index I. This index specifies a pair of domains $\mathcal{D}_I^0, \mathcal{D}_I^1$ and a pair of functions f_I^0, f_I^1. The domain sampling algorithm D is given an index I and a bit σ as input and outputs a random element from the domain $\mathcal{D}_I^\sigma$. Finally, given input $I, \sigma, x \in \mathcal{D}_I^\sigma$, the evaluation function F outputs the value of the function f_I^σ for input x. Given these definitions, they define the conditions of a claw free collection as follows.

Definition 2. *Claw-free Collections [150, 68]: A three-tuple of algorithms (G, D, F) is a claw-free collection if the following conditions hold.*

1. **Easy to evaluate:** *Both the index selecting algorithm G and the domain sampling algorithm D are probabilistic polynomial-time, while the evaluating algorithm F is deterministic polynomial-time.*

2. **Identical range distribution:** *Let $f_I^b(x)$ be the output of F given input (b, I, x). For any $I \in range(G)$, the random variables are identically distributed.*

3. **Hard to form claws:** *For every non-uniform probabilistic polynomial-time algorithm A, every polynomial $p(\cdot)$, and every sufficiently large n, it is true that $Pr[I \leftarrow G(1^n); (x, y) \leftarrow A(I) : f_I^0(x) = f_I^1(y)] < \frac{1}{p(n)}$.*

Given this definition, shelat and Shen build their scheme on a modified construction which adds in a malleability property, and focuses on a special case where $\mathcal{D}_I^0 = \mathcal{D}_I^1$ and this domain forms a group. This allows the consistency of the inputs to be checked with a witness-indistinguishable proof.

Definition 3. *Malleable Claw-Free Collection [150]: A four-tuple of algorithms (G, D, F, R) is a malleable claw-free collection if the following conditions hold:*

1. **A subset of claw-fee collections:** *(G, D, F) is a claw-free collection, and the range of D and F form groups, denoted by $(\mathbb{G}_1, \star)$ and $(\mathbb{G}_2, \diamond)$ respectively.*

2. **Uniform domain sampling:** *For any I in the range of G, random variable $D(0, I)$ and $D(1, I)$ are uniform over $\mathbb{G}_1$, and denoted by $D(I)$ for simplicity.*

3. **Malleability:** *$R : \mathbb{G}_1 \to \mathbb{G}_2$ runs in polynomial time, and for $b \in \{0, 1\}$, any I in the range of G, and any $m_1, m_2 \in \mathbb{G}_1$, $f_I^b(m_1 \star m_2) = f_I^b(m_1) \diamond R_I(m_2)$.*

Our implementation of a malleable claw-free collection uses the same construction as Kreuter et al. [103], built on the discrete logarithm assumption.

3.2.4 Model and Definitions

The work of Kamara et al. [92] presents a definition of security based on the ideal-model/real-model security definitions common in secure multiparty computation. Because their definition formalizes the idea of a non-colluding cloud, we apply their definitions to our protocol for the two-party case in particular. We summarize their definitions below.

Real-model execution. The protocol takes place between two parties (P_1, P_2) executing the protocol and a server P_3, where each of the executing parties (P_1, P_2) are provided input x_i, auxiliary input z_i, and random coins r_i for $i = 1, 2$. The server P_3 is provided only auxiliary input z_3 and random coins r_3. In the execution, there exists some subset of independent parties $(A_1, \cdots, A_m), m \leq 3$ that are malicious adversaries (we preserve this definition from Kamara et al. [92] but prove security when $m = 1$. This provides equivalent security to the Salus framework with two participants and the Cloud). Each adversary corrupts one executing party and does not share information with other adversaries. For

all honest parties, let OUT_i be its output, and for corrupted parties let OUT_i be its view of the protocol execution. The i^{th} partial output of a real execution is defined as:

$$REAL^{(i)}(k, x; r) = \{OUT_j : j \in H\} \cup OUT_i$$

where H is the set of honest parties, k is a computational security parameter, x is the set of all player inputs x_i and z_i for $i = 1, 2, 3$, and r is all random coins of all players.

Ideal-model execution. In the ideal model, the setup of participants is the same except that all parties are interacting with a trusted party that evaluates the function. All parties are provided inputs x_i, auxiliary input z_i, and random coins r_i. If a party is semi-honest, it provides its actual inputs to the trusted party, while if the party is malicious and non-colluding, it provides arbitrary input values. In the case of the server P_3, this means simply providing its auxiliary input and random coins, as no input is provided to the function being evaluated. Once the function is evaluated by the trusted third party, it returns the result to the parties P_1 and P_2, while the server P_3 does not receive the output. If a party aborts early or sends no input, the trusted party immediately aborts. For all honest parties, let OUT_i be the output given to P_i by the trusted party, and for corrupted parties let OUT_i be some value output by P_i. The *ith* partial output of an ideal execution in the presence of some set of independent simulators is defined as:

$$IDEAL^{(i)}(k, x; r) = \{OUT_j : j \in H\} \cup OUT_i$$

where H is the set of honest parties, k is a computational security parameter, x is the set of all player inputs x_i and z_i for $i = 1, 2, 3$, and r is all random coins of all players. In this model, the formal definition of security is as follows:

Definition 4. *A protocol securely computes a function f if there exists a set of probabilistic polynomial-time (PPT) simulators $\{Sim_i\}_{i=1,2,3}$ such that for all PPT adversaries $(A_1, \cdots, A_3)$, x, and for all $i = 1, 2, 3$:*

$$\{REAL^{(i)}(k, x; r)\}_{k \in N} \overset{c}{\approx} \{IDEAL^{(i)}(k, x; r)\}_{k \in N}$$

Where r is random and uniform.

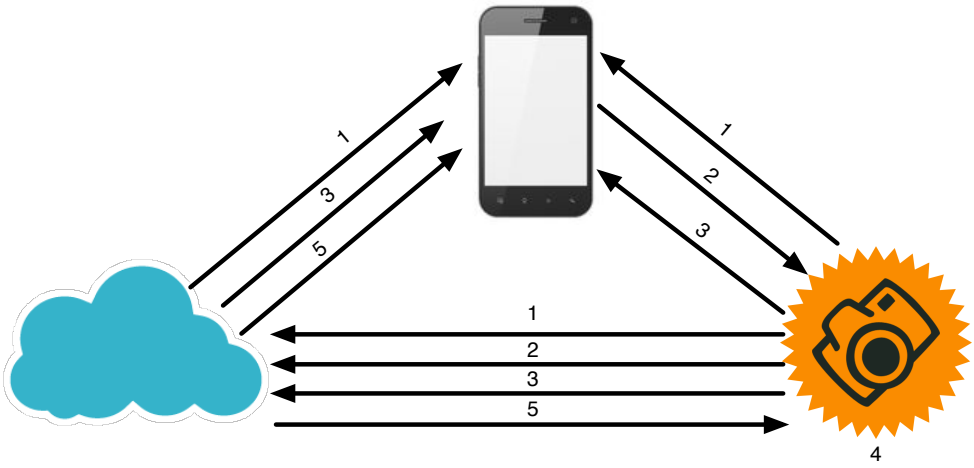


Figure 1: The complete evaluation outsourcing protocol.

Finally, we include the following lemma from Kamara et al. [91] which is used to prove the security of our protocols.

Lemma 1. *If a multi-party protocol between n parties* $(P_1, ..., P_n)$ *securely computes* f *in the presence of (1) independent and semi-honest adversaries and (2) a malicious* $\mathcal{A}_i$ *and honest* $\{\mathcal{A}_j\}_{j \neq i}$*; then it is also secure in the presence of an adversary* $\mathcal{A}_i$ *that is non-cooperative with respect to all other semi-honest adversaries.*

3.3 Protocol

3.3.1 Participants

Our protocols reference three different entities:

Mobile: The evaluating party, called Mobile, is assumed to be a mobile device that is participating in a secure two-party computation.

Application: The party generating the garbled circuit, called Application, is an application- or web- server that is the second party participating with Mobile in the secure computation.

Cloud: The proxy, called Cloud, is a third party that is performing heavy computation on behalf of Mobile, but is not trusted to know her input or the function output.

3.3.2 Protocol Phase Summary

Our protocol can be divided into five phases, illustrated in Figure 14. Given a circuit generator Application, and an evaluating mobile device Mobile, the protocol can be summarized as follows:

- Phase 1: Application generates a number of garbled circuits, some of which will be checked, others will be evaluated. After Application commits to the circuits, Mobile and Application use a fair coin toss protocol to select which circuits will be checked or evaluated. For the check circuits, Application sends the random seeds used to generate the circuits to Cloud and the hashes of each circuit to Mobile. These are checked to ensure that Application has not constructed a circuit that is corrupted or deviates from the agreed-upon function.

- Phase 2: Mobile sends her inputs to Application via an outsourced oblivious transfer. Application then sends the corresponding garbled inputs to Cloud. This allows Cloud to receive Mobile's garbled inputs without Application or Cloud ever learning her true inputs.

- Phase 3: Application sends his garbled inputs to Cloud, which verifies that they are consistent for each evaluation circuit. This prevents Application from providing different inputs to different evaluation circuits.

- Phase 4: Cloud evaluates the circuit given Mobile and Application's garbled inputs. Since Cloud only sees garbled values during the evaluation of the circuit, it never learns anything about either party's input or output. Since both output values are blinded with one-time pads, they remain private even when Cloud takes a majority vote.

- Phase 5: Cloud sends the encrypted output values to Mobile and Application, who are guaranteed its authenticity through the use of commitments and zero-knowledge proofs.

3.3.3 Outsourced Protocol

Common inputs: a function $f(x, y)$ that is to be securely computed; a malleable claw-free collection $(G_{CLW}, D_{CLW}, F_{CLW}, R_{CLW})$; a one-way, collision-resistant hash function $H : \{0,1\}^* \rightarrow \{0,1\}^k$; a primitive 1-out-of-2 oblivious transfer protocol; two commitment schemes: a perfectly binding commitment scheme $com_B(key, message)$ and the output commitment scheme $com_O(key, message)$ by Kiraz and Schoenmakers [98]; and the following security parameters: a statistical security parameter for the number of circuits built λ, a computational security parameter k, and the number of encoding bits ℓ for each of MOBILE'S input wires.

Private inputs: The generating party APPLICATION provides his input to the function a and a random string of bits a_r that is the length of the output string. The evaluating party MOBILE provides her input to the function m and a random string of bits m_r that is the length of the output string. Assume without loss of generality that all input and output strings are of length n.

Output: The protocol outputs separate private values fm for MOBILE and fa for APPLICATION.

Phase 1: Circuit generation and checking

1. *Circuit preparation:* Before beginning the protocol, both parties agree upon a circuit representation of the function $f(m, a)$, where the outputs of the function may be defined separately for MOBILE and APPLICATION as $f_M(m, a)$ and $f_A(m, a)$. The circuit must also meet the following requirements:

   (a) Additional XOR gates must be added such that APPLICATION'S output is set to $fa = f_A(m, a) \oplus a_r$ and MOBILE'S output is set to $fm = f_M(m, a) \oplus m_r$.

   (b) For each of MOBILE'S input bits, the input wire w_i is split into ℓ different input wires $w_{j,i}$ such that $w_i = w_{1,i} \oplus w_{2,i} \oplus \cdots \oplus w_{\ell,i}$ following the input encoding scheme by Lindell and Pinkas [112]. This prevents APPLICATION from correlating a selective failure attack with any of MOBILE'S input bit values.

2. *Circuit garbling:* APPLICATION first selects and malleable claw-free collection as index $I = G(1^k)$ and sends I to MOBILE. APPLICATION then takes the circuit C and creates λ independent garbled versions of C, called $GC_1 \cdots GC_\lambda$, using random coins $rc_1 \cdots rc_\lambda$ and the garbling technique implemented by Kreuter et al. [103]. For APPLICATION's j^{th} input wire on the i^{th} circuit, APPLICATION associates the value $H(\beta_{b,j,i})$ with the input value b, where $\beta_{b,j,i} = F_{CLW}(b, I, \alpha_{b,j,i})$. For MOBILE's j^{th} input wire, APPLICATION associates the value $H(\delta_{b,j,i})$ with the input value b, where $\delta_{b,j,i} = F_{CLW}(b, I, \gamma_{b,j,i})$. All the values $\alpha_{b,j,i}$ and $\gamma_{b,j,i}$ for $b = \{0,1\}, j = 1 \cdots n, i = 1 \cdots \lambda$ are selected randomly from the domain of the claw-free pair using D.

3. *Circuit commitment:* APPLICATION generates commitments for all circuits by hashing $H(GC_i) = HC_i$ for $i = 1 \cdots \lambda$. APPLICATION sends these hashes to MOBILE. In addition, for every output wire $w_{b,j,i}$ for $b = \{0,1\}$, $j = 1 \cdots n$ and $i = 1 \cdots \lambda$, APPLICATION generates commitments $CO_{j,i} = com_B(ck_{j,i}, (H(w_{0,j,i}), H(w_{1,j,i})))$ using commitment keys $ck_{j,i}$ for $j = 1 \cdots n$ and $i = 1 \cdots \lambda$ and sends them to both MOBILE and the Cloud.

4. *Input label commitment:* APPLICATION commits to MOBILE's garbled input wires as follows: for each generated circuit $i = 1 \cdots \lambda$ and each of MOBILE's input wires $j = 1 \cdots \ell \cdot n$, APPLICATION creates a pair of commitment keys $mk_{0,j,i}, mk_{1,j,i}$ and commits to the input wire label seeds $\delta_{0,j,i}$ and $\delta_{1,j,i}$ as $CM_{b,j,i} = com_B(mk_{b,j,i}, \delta_{b,j,i})$. For each of MOBILE's input wires $j = 1 \cdots \ell \cdot n$, APPLICATION randomly permutes the commitments within the pair $CM_{0,j,i}, CM_{1,j,i}$ across every $i = 1 \cdots \lambda$. This prevents CLOUD from correlating the location of the commitment with MOBILE's input value during the OOT phase.

   In the same fashion, APPLICATION commits to his own input wires: for each circuit $i = 1 \cdots \lambda$ and each of his input wires $j = 1 \cdots n$, APPLICATION creates commitment keys $ak_{0,j,i}, ak_{1,j,i}$ and commits to the input wire label seeds $\beta_{0,j,i}$ and $\beta_{1,j,i}$ as $CA_{b,j,i} = com_B(ak_{b,j,i}, \beta_{b,j,i})$, permuting each pair $CA_{0,j,i}, CA_{1,j,i}$ as above.

5. *Cut and choose:* MOBILE and APPLICATION then run a fair coin toss protocol to agree

on a set of circuits that will be evaluated, while the remaining circuits will be checked. The coin toss generates a set of indices $Chk \subset \{1, \cdots, \lambda\}$ such that $|Chk| = \frac{3}{5}\lambda$, as in shelat and Shen's cut-and-choose protocol [150]. The remaining indices are placed in the set Evl for evaluation, where $|Evl| = e = \frac{2}{5}\lambda$. For every $i \in Chk$, Application sends rc_i and the values $[\alpha_{b,1,i}, \cdots, \alpha_{b,n,i}]$ and $[\gamma_{b,1,i}, \cdots, \gamma_{b,\ell \cdot n,i}]$ for $b = \{0, 1\}$ to the Cloud. Application also sends all commitment keys $ck_{j,i}$ for $j = 1 \cdots n$ and $i \in Chk$ to Cloud. Finally, Application sends the commitment keys $mk_{b,j,i}$ and $ak_{b,j,i}$ for $b = \{0, 1\}$, $i \in Chk$, and $j = 1 \cdots \ell \cdot n$ to Cloud. Cloud then garbles C using rc_i for $i \in Chk$, producing the set of garbled circuits GC'_i for $i \in Chk$. For each $i \in Chk$, Cloud then hashes each check circuit $H(GC'_i) = HC'_i$ and checks that:

- each commitment $CO_{j,i}$ for $j = 1 \cdots n$ is well formed
- the value $H(\beta_{b,j,i})$ is associated with the input value b for Application's j^{th} input wire
- the value $H(\delta_{b,j,i})$ is associated with the input value b for Mobile's j^{th} input wire
- for every bit value b and input wire j, the values committed in $CM_{b,j,i}$ and $CA_{b,j,i}$ are correct

If any of these checks fail, Cloud immediately aborts. Otherwise, it sends the hash values HC'_i for $i \in Chk$ to Mobile. For every $i \in Chk$, Mobile checks if $HC_i = HC'_i$ to ensure that the circuits were generated correctly. If any of the hash comparisons fail, Mobile aborts.

Phase 2: Outsourced Oblivious Transfer (OOT)

1. *Input encoding:* For every bit $j = 1 \cdots n$ in her input m, Mobile sets encoded input em_j as a random string of length ℓ such that $em_{1,j} \oplus em_{2,j} \oplus \cdots \oplus em_{\ell,j} = m_j$ for each bit in m. This new encoded input string em is of length $\ell \cdot n$.

2. *OT setup:* Mobile initializes an $\ell \cdot n \times k$ matrix T with uniformly random bit values, while Application initializes a random bit vector s of length k. See Figure 2 for a

Inputs: MOBILE has a string of encoded input bits em of length $\ell \cdot n$ and APPLICATION has pairs of input values $(x_{0,j}, x_{1,j})$ for $j = 1 \cdots \ell \cdot n$.

1. **Setup:** MOBILE generates random matrix T of size $\ell \cdot n \times k$, APPLICATION generates random string s of length k.
2. **Primitive OT:** MOBILE and APPLICATION execute k 1-out-of-2 oblivious transfers with MOBILE inputting $(T^i, T^i \oplus em)$ and APPLICATION inputting selection bits s (T^i denotes the i^{th} column of the T matrix). APPLICATION sets the resulting columns as matrix Q.
3. **Permuting the output:** MOBILE generates random string p of length $\ell \cdot n$ and sends it to APPLICATION.
4. **Encrypting the output:** APPLICATION sets the encrypted output pairs $y_{0,j}, y_{1,j}$ where $y_{b,j} = x_{b,j} \oplus H_1(j, Q_j \oplus (b \cdot s))$ (Q_j denotes the j^{th} row of the matrix Q and $(b \cdot s)$ denotes component-wise multiplication by b).
5. **Permuting the outputs:** APPLICATION permutes the encrypted output pairs using p as $y_{0 \oplus p_j, j}, y_{1 \oplus p_j, j}$ and sends the resulting set of pairs Y to CLOUD.
6. **Decrypting the output:** MOBILE sends $h = em \oplus p$ and T to CLOUD. CLOUD recovers $x_{em_j, j} = y_{h_j, j} \oplus H_1(j, T_j)$ for $j = 1 \cdots \ell \cdot n$ (T_j denotes the j^{th} row of the T matrix).

Figure 2: The Outsourced Oblivious Transfer protocol

more concise view.

3. *Primitive OT operations:* With MOBILE as the sender and APPLICATION as the chooser, the parties initiate k 1-out-of-2 oblivious transfers. MOBILE'S input to the i^{th} instance of the OT is the pair $(T^i, T^i \oplus ea)$ where T^i is the i^{th} column of T, while APPLICATION'S input is the i^{th} selection bit from the vector s. APPLICATION organizes the k selected columns as a new matrix Q.

4. *Permuting the selections:* MOBILE generates a random bit string p of length $\ell \cdot n$, which she sends to APPLICATION.

5. *Encrypting the commitment keys:* APPLICATION generates a matrix of keys that will open the committed garbled input values and proofs of consistency as follows: for

MOBILE's j^{th} input bit, APPLICATION creates a pair $(x_{0,j}, x_{1,j})$, where

$$x_{b,j} = [mk_{b,j,Evl_1}, mk_{b,j,Evl_2}, \cdots, mk_{b,j,Evl_e}] || [\gamma_{b_j,j,Evl_2} \star (\gamma_{b_j,j,Evl_1})^{-1},$$
$$\gamma_{b_j,j,Evl_3} \star (\gamma_{b_j,j,Evl_1})^{-1}, \cdots, \gamma_{b_j,j,Evl_e} \star (\gamma_{b_j,j,Evl_1})^{-1}]$$

and Evl_i denotes the i^{th} index in the set of evaluation circuits. For $j = 1 \cdots \ell \cdot n$, APPLICATION prepares $(y_{0,j}, y_{1,j})$ where $y_{b,j} = x_{b,j} \oplus H(j, Q_j \oplus (b \cdot s))$. Here, Q_j denotes the j^{th} row in the Q matrix and $(b \cdot s)$ denotes component-wise multiplication by the bit b. APPLICATION permutes the entries using MOBILE's permutation vector as $(y_{0 \oplus p_j,j}, y_{1 \oplus p_j,j})$. APPLICATION sends this permuted set of ciphertexts Y to CLOUD.

6. *Receiving* MOBILE's *garbled inputs:* MOBILE blinds her input as $h = em \oplus p$ and sends h and T to CLOUD. CLOUD recovers the commitment keys and consistency proofs $x_{b,j} = y_{h_j,j} \oplus H(j, T_j)$ for $j = 1 \cdots \ell \cdot n$. Here, h_j denotes the j^{th} bit of the string h and T_j denotes the j^{th} row in the T matrix. Since for every $j \in Evl$, CLOUD only has the commitment key for the b garbled value (not the $b \oplus 1$ garbled value), CLOUD can correctly decommit only the garbled labels corresponding to MOBILE's input bits.

7. *Verifying consistency across* MOBILE's *inputs:* Given the decommitted values $[\delta_{b,1,i}, \cdots, \delta_{b,\ell \cdot n,i}]$ and the modified pre images $[\gamma_{b_j,j,Evl_2} \star (\gamma_{b_j,j,Evl_1})^{-1}, \gamma_{b_j,j,Evl_3} \star (\gamma_{b_j,j,Evl_1})^{-1}, \cdots, \gamma_{b_j,j,Evl_e} \star (\gamma_{b_j,j,Evl_1})^{-1}]$, CLOUD checks that:

$$\delta_{b_j,j,i} = \delta_{b_j,j,Evl_1} \diamond R_{CLW}(I, \gamma_{b_j,j,i} \star (\gamma_{b_j,j,Evl_1})^{-1})$$

for $i = 2 \cdots e$. If any of these checks fails, CLOUD aborts the protocol. Otherwise, CLOUD now has garbled versions of MOBILE's inputs gm_i for $i \in Evl$.

Phase 3: Generator input consistency check

1. *Delivering inputs:* APPLICATION decommits the hash seeds for each of his garbled input values by sending $ak_{a_i,j,i}$ to CLOUD for $j = 1 \cdots n$ and $i \in Evl$. CLOUD recovers the hash seeds $[\beta_{a_1,1,i}, \beta_{a_2,2,i}, \cdots, \beta_{a_n,n,i}]$ for every evaluation circuit $i \in Evl$ and forwards a copy of these values to MOBILE. APPLICATION then proves the consistency of his inputs by sending the modified preimages $[\alpha_{a_j,j,Evl_2} \star (\alpha_{a_j,j,Evl_1})^{-1}, \alpha_{a_j,j,Evl_3} \star$

$(\alpha_{a_j,j,Evl_1})^{-1}, \cdots, \alpha_{a_j,j,Evl_e} \star (\alpha_{a_j,j,Evl_1})^{-1}]$ such that $F_{CLW}(a_i, I, \alpha_{a_i,j,i}) = \beta_{a_i,j,i}$ for $j = 1 \cdots n$ and $i \in Evl$ such that GC_i was generated with the claw-free function pair indexed at I.

2. *Check consistency:* Mobile then checks that all the hash seeds were generated by the same function by checking if:

$$\beta_{a_j,j,i} = \beta_{a_j,j,Evl_1} \diamond R_{CLW}(I, \alpha_{a_j,j,i} \star (\alpha_{a_j,j,Evl_1})^{-1})$$

for $i = 2 \cdots e$. If any of these checks fails, Mobile aborts the protocol. Otherwise, Cloud now has garbled versions of Application's inputs ga_i for $i \in Evl$.

Phase 4: Circuit evaluation

1. *Evaluating the circuit:* For each evaluation circuit, Cloud evaluates $GC_i(gm_i, ga_i)$ for $i \in Evl$ in the pipelined manner described by Kreuter et al. [103]. Each circuit produces two garbled output strings, (gfm_i, gfa_i).

2. *Checking the evaluation circuits:* Once these output have been computed, Cloud hashes each evaluation circuit as $H(GC_i) = HC'_i$ for $i \in Evl$ and sends these hash values to Mobile (this hash is performed in parallel with the previous step). Mobile checks that for every $i, HC_i = HC'_i$. If any of these checks do not pass, Mobile aborts the protocol.

Phase 5: Output check and delivery

1. *Committing the outputs:* Cloud then generates random commitment keys km_i, ka_i and commits the output values to their respective parties according to the commitment scheme defined by Kiraz and Schoenmakers [98], generating $CM_{j,i} = com_O(km_{j,i}, gfm_{j,i})$ and $CA_{j,i} = com_O(ka_{j,i}, gfa_{j,i})$ for $j = 1 \cdots n$ and $i = 1 \cdots e$. Cloud then sends all CM to Mobile and CA to Application.

2. *Selection of majority output:* Application opens the commitments $CO_{j,i}$ for $j = 1 \cdots n$ and $i = 1 \cdots e$ for both Mobile and Cloud. These commitments contain the mappings from the hash of each garbled output wire $H(w_{b,j,i})$ to real output values

$b_{j,i}$ for $j = 1 \cdots n$ and $i = 1 \cdots e$ (note that the result of computation is still blinded from CLOUD by m_r and a_r). CLOUD selects a circuit index maj such that the output of that circuit matches the majority of outputs for both MOBILE and APPLICATION. That is, $fm_{maj} = fm_i$ and $fa_{maj} = fa_i$ for i in a set of indices IND that is of size $|IND| > \frac{e}{2}$

3. *Proof of output consistency:* Using the OR-proofs as described by Kiraz and Schoenmakers [98], CLOUD proves to APPLICATION that CB contains valid garbled output bit values based on the de-committed output values from the previous step. CLOUD then performs the same proof to MOBILE for her committed values CM. Note that these proofs guarantee the output was generated by one of the circuits, but the value maj remains hidden from both MOBILE and APPLICATION, which prevents APPLICATION from learning anything based on knowledge of how circuit maj was constructed.

4. *Output release:* CLOUD then decommits gfm_{maj} to MOBILE and gfa_{maj} to APPLICATION. Given these garbled outputs and the bit values corresponding to the hash of each output wire, MOBILE recovers her output string fm, and APPLICATION recovers his output string fa.

5. *Output decryption:* MOBILE recovers her output $f_M(m, a) = fm \oplus m_r$, while APPLICATION recovers $f_A(m, a) = fa \oplus a_r$.

3.3.4 Asymptotic Evaluation

When considering the performance of our outsourcing scheme, our goal is to optimize the workload on the mobile device. Because this device will always be more limited in computing capability than CLOUD, we need to minimize the overhead at this bottleneck. The dominating operations on the mobile device can be summarized in three categories: the OOT protocol, which requires k oblivious transfers and $O(|m|)$ symmetric operations to generate the matrix T; the input consistency check, which requires $O(\lambda|b|)$ group operations; and the output verification proof, which requires $O(\lambda|f_M(m, a)|)$ additive homomorphic operations (i.e., Diffie-Hellman group operations) in the OR-proof and $|f_M(m, a)|$ XOR operations to

Table 1: Asymptotic analysis of the operations required on the mobile device for each outsourcing protocol. Here, SYM is symmetric cryptographic operations, GROUP is group algebraic operations, and OT is oblivious transfers. Recall that k is the security parameter, λ is the number of circuits generated, m is MOBILE's input, and a is APPLICATION's input.

Protocol	SYM	GROUP	OT
CMTB	$O(\|m\|)$	$O(\lambda(\|a\| + \|f_M(m,a)\|))$	k
Salus	$O(\lambda(\|m\| + \|a\| + \|f_M(m,a)\|))$	-	-

decrypt the output (this single operation is dominated by the $O(\lambda|f_M(m,a)|)$ operations in the OR-proof and is omitted from our total). These operations produce an overall complexity of $O(\lambda(|b| + |f_M(m,a)|) + |m| + k)$. When compared to the outsourcing technique developed by Kamara et al. [91, 92], which incurs an overhead of $O(\lambda(|m| + |a| + |f(m,a)|))$ on the mobile device, our protocol has comparable asymptotic performance overall, and better asymptotic performance when the mobile input m is large. However, because Salus is implemented exclusively with symmetric cryptographic operations and our protocol requires several group operations, our improved asymptotic performance would not produce real-time performance improvements unless the size of m is very large. These values are summarized in Table 3.

3.4 Security Guarantees

In this section, we provide a summary of the security mechanisms used in our protocol and an informal discussion of the security guarantees of our outsourced oblivious transfer construction. While all of the basic consistency checks already exist in previous work, our protocol demonstrates how these checks can be modified to allow for secure computation in the outsourced model. In combination with our novel outsourced oblivious transfer protocol, this construction provides a significant step towards practical SFE on mobile devices that is secure against malicious adversaries.

Recall from Section 3.2 that there are generally four security concerns when evaluating garbled circuits in the malicious setting. To solve the problem of malicious circuit generation, we apply the random seed check variety of cut-and-choose developed by Goyal et al. [72]. To solve the problem of selective failure attacks, we employ the input encoding

technique developed by Lindell and Pinkas [112]. To prevent an adversary from using inconsistent inputs across evaluation circuits, we employ the witness-indistinguishable proofs from shelat and Shen [150]. Finally, to ensure the majority output value is selected and not tampered with, we use the XOR-and-prove technique from Kiraz and Schoenmakers [98] as implemented by Kreuter et al. [103]. In combination with the standard semi-honest security guarantees of Yao garbled circuits, these security extensions secure our scheme in the malicious security model.

3.4.1 Garbled Circuit Generation

To ensure the evaluated circuits are generated honestly, we require two properties. First, we limit the generator Application's ability to trick Mobile into evaluating a corrupted circuit using a cut-and-choose technique similar to a typical, two-party garbled circuit evaluation. Second, we ensure that a lazy Cloud attempting to conserve system resources cannot bypass the circuit checking step without being discovered. We call this Cloud's "proof-of-work".

Claim 1. *Security: Assuming that the hash function $H(\cdot)$ is a one-way, collision-resistant hash and that the commitment scheme used is fully binding, then the generator* Application *has at best a $2^{-0.32\lambda}$ probability of tricking* Mobile *into evaluating a majority of corrupted circuits, where λ is the number of circuits generated.*

shelat and Shen [150] perform a rigorous analysis of the optimal cut-and-choose strategy for evaluating garbled circuits in the malicious setting. Given that the generator prepares λ circuits before the cut-and-choose, their protocol sets the number of check circuits to $\frac{3\lambda}{5}$ rather than $\frac{\lambda}{2}$, which is found in previous schemes. By their analysis, this provides a security level of $2^{-0.32\lambda}$. Since our scheme is built on the implementation by Kreuter et al. [103], which uses the same cut-and-choose parameter as shelat and Shen, our garbled circuit check also provides a security parameter of $2^{-0.32\lambda}$ which is non-polynomial and negligible for large λ.

Claim 2. *Proof-of-work: Assuming the hash function H is one-way and collision resistant,*

CLOUD *has a negligible probability in the security parameter k of producing a check hash that passes the seed check without actually generating the check circuit.*

As previously stated, before the circuit check begins the generator APPLICATION sends the evaluator MOBILE λ hashed circuit values $H(GC_i)$. Once the evaluation circuits are selected, CLOUD must generate λ circuits and hash them into check hashes $H(GC'_i)$. We assume that CLOUD does not collude with the generator (i.e. share any of the hash values sent to MOBILE). If CLOUD attempts to skip the generation of the check circuits, it must generate hash values $H'_i = H_i$ for $i \in Chk$. Based on security guarantees of the hash with output length k, CLOUD has a negligible probability of correctly generating these hash values.

3.4.2 Validity of Evaluator Inputs

To assure that the generator cannot learn anything about the evaluator's inputs by corrupting the garbled values sent during the OT, we employ the random input encoding technique by Lindell and Pinkas [112], which is built into the implementation by Kreuter et al. [103]. This technique allows the evaluator to encode each input bit as the XOR of a set of input bits. Thus, if the generator corrupts one of those input bits as in a selective failure attack, it reveals essentially nothing about the evaluator's true input. Additionally, we use the commitment technique employed by Kreuter et al. [103] to ensure that APPLICATION cannot swap garbled input wire labels between the zero and one value. To accomplish this, the generator commits to the wire labels before the cut-and-choose. During the cut and choose, the input labels for the check circuits are opened to ensure that they correspond to only one value across all circuits. Then, during the OOT, the commitment keys for the labels that will be evaluated are sent instead of the wire labels themselves. Because our protocol implements this technique directly from the literature, we do not make any additional claims of security.

3.4.3 Input Consistency

The security of our input consistency check is based on two schemes, one for the evaluator's input and one for the generator's input. To assure the evaluator's inputs are consistent

across circuits, we use the approach from Lindell and Pinkas [112], which is built into the implementation of Kreuter et al. [103]. Since the evaluator only performs one oblivious transfer for all the evaluation circuits, her received garbled inputs will all represent her input to the OT.

To assure that the generator's inputs are consistent, we employ the malleable claw-free collection approach from shelat and Shen [150]. However, we modify the zero-knowledge proof to provide some guarantee that CLOUD actually possesses well-formed inputs:

Claim 3. *Assuming the witness-indistinguishable proof used in the malleable claw-free collection input check catches inconsistent inputs except for a negligible probability, the generator in our protocol cannot trick the evaluator into using different inputs for different evaluation circuits with greater than negligible probability.*

During the witness-indistinguishable proof, the generator sends the modified pre-image values to the mobile device, while CLOUD sends the garbled input values of each evaluation circuit to the mobile device. The device then checks that all input values for each individual input wire were generated by the same function in the malleable claw-free pair. Based on the assumption that the generator and CLOUD will not collude, the probability of a malicious generator providing inconsistent modified pre-image values that match the garbled inputs possessed by CLOUD is negligible in the security parameter of the malleable claw-free pair.

3.4.4 Output Consistency

To ensure that CLOUD cannot learn either party's output or tamper with either party's output from the garbled circuit, we implement the technique of blinding and proving the garbled output values from the protocol by Kiraz and Schoenmakers [98]. The privacy and correctness of the generator's output is guaranteed based on the security of this construction in Kiraz's two-party secure function evaluation protocol. By the same proof for the generator's output remaining secure, we argue that the evaluator's output is also secure and correct. By using the same construction for both parties' outputs, we guarantee output privacy and consistency, even in the presence of a malicious CLOUD. Note that to maintain security, this construction only provides APPLICATION and MOBILE with the output

of computation, not the index of the majority evaluation circuit.

3.4.5 Outsourced Oblivious Transfer

Our outsourced oblivious transfer is an extension of a technique developed by Naor et al. [131] that allows the chooser to select entries that are forwarded to a third party rather than returned to the chooser. By combining their concept of a proxy oblivious transfer with the semi-honest OT extension by Ishai et al. [87], our outsourced oblivious transfer provides a secure OT in the malicious model. We achieve this result for four reasons:

1. First, since Mobile never sees the outputs of the OT protocol, she cannot learn anything about the garbled values held by the generator. This saves us from having to implement Ishai's extension to prevent the chooser from behaving maliciously.

2. Since Cloud sees only random garbled values and Mobile's input blinded by a one-time pad, Cloud learns nothing about Mobile's true inputs.

3. Since Application's view of the protocol is almost identical to his view in Ishai's standard extension, the same security guarantees hold (i.e., security against a malicious sender).

4. Finally, if Mobile does behave maliciously and uses inconsistent inputs to the primitive OT phase, there is a negligible probability that those values will hash to the correct one-time pad keys for recovering *either* commitment key, which will prevent Cloud from de-committing the garbled input values.

It is important to note that this particular application of the OOT allows for this efficiency gain since the evaluation of the garbled circuit will fail if Mobile behaves maliciously. By applying the maliciously secure extension by Ishai et al. [87], this primitive could be applied generally as an oblivious transfer primitive that is secure in the malicious model. Further discussion and analysis of this general application is outside the scope of this work.

3.5 Proof of Security

We formally prove the security of our protocol with the following theorem, which gives security guarantees identical to the Salus protocol by Kamara et al. [92].

Theorem 1. *The outsourced two-party SFE protocol securely computes a function $f(a, b)$ in the following two corruption scenarios: (1)* Cloud *is malicious and non-cooperative with respect to the rest of the parties, while all other parties are semi-honest, (2)* Cloud *is semi-honest and exactly one other party is malicious.*

Proof. To demonstrate that:

$$\{REAL^{(i)}(k, x; r)\}_{k \in N} \overset{c}{\approx} \{IDEAL^{(i)}(k, x; r)\}_{k \in N}$$

for all $i \in [M, A, C]$, we consider separately the cases when Mobile, Application, and Cloud deviate from the protocol.

3.5.1 Malicious evaluator Mobile M^*

In this scenario, both Application and Cloud participate honestly in the protocol. Note that during the protocol execution, Mobile only exchanges messages with the other participants at five points: the coin-flip during the cut-and-choose, the primitive oblivious transfer, sending decryption information at the end of the OOT, checking Application's input consistency, and receiving the proof of validity and output from the garbled circuit. Thus, our simulator needs only ensure that these sections of the protocol are indistinguishable to the adversary M^* with respect to the security parameter k. Consider the following hybrid experiments and lemmas.

Simulating the coin-flip (Phase 1):

$\boldsymbol{Hybrid1^{(M)}(k, x; r)}$: This experiment is the same as $REAL^{(M)}(k, x; r)$ except that instead of running a fair coin toss protocol with M^*, the experiment chooses a random string ρ, and a coin-flipping simulator $S_{CF}(\rho, 1^k)$ produces the protocol messages that output ρ.

Lemma 2. $REAL^{(M)}(k, x; r) \stackrel{c}{\approx} Hybrid1^{(M)}(k, x; r)$

Proof. Based on the security of the fair coin toss protocol, we know that there exists a simulator $S_{CF}(\cdot, \cdot)$ such that an interaction with $S_{CF}(\cdot, \cdot)$ is indistinguishable from a real protocol interaction. Since everything else in $Hybrid1^{(M)}(k, x; r)$ is exactly the same as in $REAL^{(M)}(k, x; r)$, this proves the lemma. □

Simulating the primitive OT (Phase 2):

$\boldsymbol{Hybrid2^{(M)}(k, x; r)}$: This experiment is the same as $Hybrid1^{(M)}(k, x; r)$ except that during the Outsourced Oblivious Transfer, the experiment invokes a simulator S_{OT} to simulate the primitive oblivious transfer operation with M^*. The simulator sends M^* a random string s and receives the columns of the matrix Q^*.

Lemma 3. $Hybrid1^{(M)}(k, x; r) \stackrel{c}{\approx} Hybrid2^{(M)}(k, x; r)$

Proof. Based on the malicious security of the OT primitive, we know that there exists a simulator S_{OT} such that an interaction with this simulator is indistinguishable from a real execution of the oblivious transfer protocol. Since everything else in $Hybrid2^{(M)}(k, x; r)$ is identical to $Hybrid1^{(M)}(k, x; r)$, this proves the lemma. □

Checking the output of OOT (Phase 2):

$\boldsymbol{Hybrid3^{(M)}(k, x; r)}$: This experiment is the same as $Hybrid2^{(M)}(k, x; r)$ except that the experiment aborts if the matrix Q^* is not formed correctly (that is, if M^* used inconsistent input values em^* for any column in generating Q^*).

Lemma 4. $Hybrid2^{(M)}(k, x; r) \stackrel{c}{\approx} Hybrid3^{(M)}(k, x; r)$

Proof. Consider that in $Hybrid2^{(M)}(k, x; r)$, if for some value of i, M^* sends the column value $T^i \oplus em'$ for some $em' \neq em^*$ such that the i^{th} bit is b in em^* and $b \oplus 1$ in em'. Then for every row in Q^*, the i^{th} bit will be encrypted in the $b \oplus 1$ entry. However, when M^* sends the value $em^* \oplus p^*$ to CLOUD for decryption, when CLOUD decrypts the i^{th}

choice, it will decrypt the $b \oplus 1$ entry instead of the b entry, which will yield an invalid decryption with probability $1 - \epsilon$ for a negligible value of ϵ. Since, with high probability, this decryption is not a valid commitment key, the garbled input values will not decommit properly and CLOUD will abort. In $Hybrid3^{(M)}(k, x; r)$, since the experiment observes the messages Q^*, p^*, and $em^* \oplus p^*$, it can recover em^* and check Q^* for consistency, aborting if an inconsistency is found. □

Simulating consistency check and substituting inputs (Phase 3):

***Hybrid4*$^{(M)}(k, x; r)$**: This experiment is the same as $Hybrid3^{(M)}(k, x; r)$ except that the experiment provides a string of $2 \cdot n$ zeros, denoted $\{0\}^{2 \cdot n}$, during the consistency check to replace APPLICATION'S input a.

Lemma 5. $Hybrid3^{(M)}(k, x; r) \overset{c}{\approx} Hybrid4^{(M)}(k, x; r)$

Proof. Here we cite Lemma 5 from the proof of shelat and Shen's scheme [150]. Since the messages sent in our scheme are identical to theirs in content, we simply change the entity sending the message in the experiment and the lemma still holds. □

Simulating the output proof (Phase 5):

***Hybrid5*$^{(M)}(k, x; r)$**: This experiment is the same as $Hybrid4^{(M)}(k, x; r)$ except that instead of returning the output of the circuit, the experiment provides M^* with the result sent from the trusted external oracle.

Lemma 6. $Hybrid4^{(M)}(k, x; r) \overset{c}{\approx} Hybrid5^{(M)}(k, x; r)$

Proof. Based on the security of the garbled circuit construction being used, the trusted third party output and the circuit output will be indistinguishable when provided with M^*s input em^*, which the experiment can recover because of the change made in $Hybrid3^{(M)}(k, x; r)$. In addition, since the experiment can observe the random seeds used to construct the proofs of output consistency used when generating the evaluation circuits, the experiment can reproduce valid proofs of consistency for the output value $f_M(m^*, a)$ provided by the oracle.

Based on the security proofs of these consistency checks by Kiraz and Schoenmakers [98], indistinguishability holds. □

Lemma 7. *$Hybrid5^{(M)}(k, x; r)$ runs in polynomial time.*

Proof. Since M^* is strictly a polynomial-time adversary and all of the operations in the *REAL* protocol are polynomial time, most of the operations performed can be summarized into a runtime $p(k)$. The two simulators S_{OT} and $S_{CF}(\cdot, \cdot)$ are assumed to be polynomial in runtime since they are computationally secure simulators for their respective roles. Since they are both only executed once, the total running time can be expressed as $p(k) + r_{OT} + r_{CF}$, where r_{OT} is the runtime of the polynomial simulator S_{OT} and r_{CF} is the runtime of the polynomial simulator $S_{CF}(\cdot, \cdot)$. Since all of these individual components are polynomial, the total runtime is also polynomial. □

$Hybrid5^{(M)}(k, x; r)$ is exactly the experiment run by the simulator S^M in the ideal world. Should M^* ever abort the protocol, the simulator S^M will forward the abort to the trusted third party. Otherwise, it will follow $Hybrid5^{(M)}(k, x; r)$, controlling APPLICATION and CLOUD, and outputs whatever M^* outputs. By Lemma 2-7, this simulator proves Theorem 1 when MOBILE is malicious.

3.5.2 Malicious generator Application A^*

In this scenario, both MOBILE and CLOUD participate honestly in the protocol. Note that in the protocol, the generator exchanges messages with both parties at five critical points: circuit cut-and-choose, the primitive OT, the OOT result delivery, the input consistency check, and the output proof of integrity and delivery. Consider the following hybrid experiments and lemmas.

Simulating the cut-and-choose (Phase 1):

***Hybrid*$\mathbf{1^{(A)}(k, x; r)}$**: This experiment is the same as $REAL^{(A)}(k, x; r)$ except that if A^* successfully passes the first cut-and-choose test, the experiment repeatedly rewinds A^*,

repeating the coin flip protocol and the verification of the circuit hashes and commitments until A^* passes for a second time. Let Chk_i be the set of check circuit indices for the i^{th} successful cut-and-choose. If $Chk_1 = Chk_2$, then the experiment aborts.

Lemma 8. $REAL^{(A)}(k, x; r) \overset{c}{\approx} Hybrid1^{(A)}(k, x; r)$

Proof. Here we cite Lemma 8 from shelat and Shen's protocol proof [150]. The idea in their proof is that if A^* never passes the cut-and-choose, then both the real and hybrid experiments will abort. However, if A^* passes once, they demonstrate that the probability of A^* passing again with the exact same set of check circuits is negligible within a polynomial number of rewinds. Since their cut-and-choose is identical to ours, indistinguishability holds in this setting. □

Checking input consistency and recovering inputs (Phase 3):

$\boldsymbol{Hybrid2^{(A)}(k, x; r)}$: This experiment is the same as $Hybrid1^{(A)}(k, x; r)$ except that the experiment recovers A^*'s input a^* during the input consistency check using the random seed recovered in $Hybrid1^{(A)}(k, x; r)$. Since the experiment is running with a set of evaluation circuits Evl_2 produced during the second successful cut-and-choose, it possesses the random coins used to generate at least one of these circuits since $Evl_1 \neq Evl_2$. If the consistency check does not pass or if A^*'s input cannot be recovered, the experiment immediately aborts.

Lemma 9. $Hybrid1^{(A)}(k, x; r) \overset{c}{\approx} Hybrid2^{(A)}(k, x; r)$

Proof. Based on the security of shelat and Shen's claw-free collections technique for checking the consistency of A^*'s inputs across evaluation circuits, then A^* can find a claw and change its input for one evaluation circuit with negligible probability. In $Hybrid2^{(A)}(k, x; r)$, since the experiment possesses at least one set of random coins rc_i that was used to generate a circuit in the set Evl_2, it can recover the input wire labels for that circuit and recover A^*'s input. If the input is invalid (i.e., does not correspond to an input label), the circuit would fail to evaluate and would generate an abort in both hybrids. Thus, indistinguishability holds. □

Simulating the output proof (Phase 5):

***Hybrid3*$^{(A)}(k, x; r)$**: This experiment is the same as $Hybrid2^{(A)}(k, x; r)$ except that during the output phase the experiment prepares the result $f(m, a^*)$ received from the trusted third party as the output instead of the output from the circuit $f(m, a^*)$. If no majority values fa' are found from the circuit, the experiment aborts. Otherwise, it uses the random coins rc_i recovered from $Hybrid1^{(A)}(k, x; r)$ to prove the validity of the output for some circuit.

Lemma 10. $Hybrid2^{(A)}(k, x; r) \stackrel{c}{\approx} Hybrid3^{(A)}(k, x; r)$

Proof. Based on the security of the garbled circuit construction being used, the trusted third party output and the circuit output will be indistinguishable when provided with A^*'s input a^*, which the experiment can recover using the random coins rc_i obtained in $Hybrid1^{(A)}(k, x; r)$. In addition, these random coins allow the experiment to construct the proofs of output consistency used when generating one of the evaluation circuits, thus the experiment can reproduce valid proofs of consistency for the output value $f_A(m, a^*)$ provided by the trusted third party. If no majority output value exists, in both hybrids the abort message would be sent. Finally, based on the security proofs of the witness indistinguishable consistency proofs developed by Kiraz and Schoenmakers [98], indistinguishability holds. □

Simulating the primitive OT (Phase 2):

***Hybrid4*$^{(A)}(k, x; r)$**: This experiment is the same as $Hybrid3^{(A)}(k, x; r)$ except that rather than run the primitive oblivious transfer with MOBILE, the experiment generates a random input string m' and a random matrix T, then runs a simulator S_{OT} with A^*, which delivers to A^* exactly one element from the pair $(T^i, T^i \oplus em')$ depending on A^*'s i^{th} selection bit.

Lemma 11. $Hybrid3^{(A)}(k, x; r) \stackrel{c}{\approx} Hybrid4^{(A)}(k, x; r)$

Proof. Based on the security of the primitive OT scheme, we know that the simulator S_{OT} exists, that it can recover A^*'s selection bits s^* from the interaction, and that an interaction with it is indistinguishable from a real execution of the OT. Since A^* cannot learn any distinguishing information from MOBILE's input, again based on the security of the OT primitive, then indistinguishability holds between the hybrid experiments. Note that the ordering of hybrids 4 and 5 is critical. If the Phase 2 hybrids were included in protocol order, the adversary A^* would receive the output $f_A(m', a^*)$ instead of $f_A(m, a^*)$, which would allow him to distinguish between the real and ideal worlds. □

Checking the output of OOT (Phase 2):

***Hybrid*$\mathbf{5^{(A)}(k, x; r)}$**: This experiment is the same as $Hybrid4^{(A)}(k, x; r)$ except that the experiment checks the validity of A^*'s output from the OOT. Since the experiment possesses T, em', and s^* (which was recovered by the oblivious transfer simulator S_{OT} in the previous hybrid), the experiment can check whether or not the encrypted set of outputs Y^* is well-formed. If it is not, the experiment immediately aborts.

Lemma 12. $Hybrid4^{(A)}(k, x; r) \overset{c}{\approx} Hybrid5^{(A)}(k, x; r)$

Proof. Recall that in $Hybrid4^{(A)}(k, x; r)$, if A^* does not format the output of the OOT correctly, CLOUD will, with probability $1 - \epsilon$ fail to recover a valid commitment key, where ϵ is negligible in the security parameter. Should this be the case, the committed garbled circuit labels will fail to decrypt properly, and CLOUD will abort the protocol. In $Hybrid4^{(A)}(k, x; r)$, since the experiment has observed the values Q, s^*, em', and Y^*, it can trivially observe if Y^* is correctly formed, and aborts if it is not. Additionally, for A^* to swap any of M's input labels in the commitments, A^* must find a claw in the claw-free collection used to generate those input labels. Based on the security of shelat and Shen's claw-free collections technique, used to check the consistency of M's input across evaluation circuits in this phase, this will only happen with a negligible probability. □

Lemma 13. $Hybrid5^{(A)}(k, x; r)$ *runs in polynomial time.*

Proof. Since all of the steps in the protocol run in expected polynomial time, the only step that must be verified is the rewinding phase. Based on Lemma 14 from shelat and Shen's proof [150], the total time for the rewinds is also polynomial in k. Thus, the composition of all steps runs in polynomial time. □

$Hybrid5^{(A)}(k, x; r)$ is exactly the experiment run by the simulator S^A in the ideal world. Should A^* ever abort the protocol, the simulator S^A will forward the abort to the trusted third party. Otherwise, it will follow $Hybrid5^{(A)}(k, x; r)$, controlling MOBILE and CLOUD, and outputs whatever A^* outputs. By Lemma 8-13, this simulator proves Theorem 1 when APPLICATION is malicious.

3.5.3 Malicious Cloud C^*

In this scenario, both MOBILE and APPLICATION participate honestly in the protocol. Note that in the protocol, CLOUD participates in checking the circuits during the cut-and-choose, decrypting MOBILE'S inputs in the OOT, forwarding APPLICATION'S inputs for consistency checking, evaluating the circuit, and proving and delivering the final output of computation. Consider the following hybrid experiments and lemmas.

Replacing inputs for the OOT (Phase 2):

$\boldsymbol{Hybrid1^{(C)}(k, x; r)}$: This experiment is the same as $REAL^{(C)}(k, x; r)$ except that during the OOT, the experiment replaces MOBILE'S input em with a string of zeros $em' = \{0\}^{2 \cdot \ell \cdot n}$. This value is then used to select garbled input values from APPLICATION in the OOT, which are then forwarded to C^* according to the protocol.

Lemma 14. $REAL^{(C)}(k, x; r) \overset{c}{\approx} Hybrid1^{(C)}(k, x; r)$

Proof. In a real execution, C^* will observe the random matrix T, the encrypted commitment keys Y, and MOBILE'S input XOR'd with the permutation string $em \oplus p$. Based on the statistical indistinguishability of a value XOR'd with a random value, $em \oplus p \overset{s}{\approx} p$ for any input value em. Since T is randomly generated in both $REAL^{(C)}(k, x; r)$ and $Hybrid1^{(C)}(k, x; r)$,

they are trivially indistinguishable. Considering the output pairs Y, half of the commitment keys (those not selected by MOBILE) will consist of the keys in $x_{b,j} \oplus H(j,s)$, which is computationally indistinguishable from random, and the keys in $x_{b,j}$ can only be recovered if C^* can find a collision with the hash value $H(j,s)$ without having APPLICATION'S random value s. The remaining keys, which can be recovered by C^*, are permuted randomly, such that their ordering is statistically indistinguishable from a random ordering. Since the commitments are also permuted randomly, the same indistinguishability holds for the ordering of the garbled input wire values. Thus, C^* cannot distinguish an execution of OOT with MOBILE'S input em and the simulator's input replacement em'. Since the rest of the protocol follows $REAL^{(C)}(k,x;r)$ exactly, this proves the lemma. □

Replacing inputs for the consistency check (Phase 3):

$\boldsymbol{Hybrid2^{(C)}(k,x;r)}$: This experiment is the same as $Hybrid1^{(C)}(k,x;r)$ except that the experiment replaces APPLICATION'S input a with all zeros $\{0\}^{2\cdot n}$. This value is then prepared and checked according to the protocol for consistency across evaluation circuits.

Lemma 15. $Hybrid1^{(C)}(k,x;r) \overset{c}{\approx} Hybrid2^{(C)}(k,x;r)$

Proof. In this hybrid, C^* observes a set of garbled input wire values from APPLICATION. Based on the security of Yao garbled circuits, observing one set of garbled input wire values is indistinguishable from observing any other set of input wire values, such that C^* cannot distinguish between the garbled input for a and the garbled input for $\{0\}^{2\cdot n}$. Since the rest of the hybrid is the same as $Hybrid1^{(C)}(k,x;r)$, this proves the lemma. □

Checking the output of the circuit (Phase 5):

$\boldsymbol{Hybrid3^{(C)}(k,x;r)}$: This experiment is the same as $Hybrid2^{(C)}(k,x;r)$ except that after the circuit is evaluated, the experiment checks that the results output by C^* matches the expected results $f_M(m',a')$ and $f_A(m',a')$. If C^* fails to produce a valid proof that the output came from the circuit or if the result does not match, the experiment immediately

aborts.

Lemma 16. $Hybrid2^{(C)}(k, x; r) \stackrel{c}{\approx} Hybrid3^{(C)}(k, x; r)$

Proof. In $Hybrid2^{(C)}(k, x; r)$, C^* can only modify the output without detection with probability $1 - \epsilon$ for some negligible probability ϵ, based on the security guarantees of Kiraz's proof scheme [98]. In $Hybrid3^{(C)}(k, x; r)$ the experiment catches C^* when it tries to change the output of the circuit with probability 1, so the distributions are computationally indistinguishable. □

Lemma 17. $Hybrid3^{(C)}(k, x; r)$ *runs in polynomial time.*

Proof. Since the protocol itself requires only polynomially many steps, the time to run $REAL^{(C)}(k, x; r)$ can be expressed as $p(k)$. The experiment also evaluates the polynomial-time function $f(\cdot, \cdot)$ over random inputs to check the output of C^*. We call this execution time $q(c)$, where c is the size of the circuit being evaluated. So, the total execution time is $p(k) + q(c)$, which is polynomial as a sum. □

$Hybrid3^{(C)}(k, x; r)$ is exactly the experiment run by the simulator S^C in the ideal world. Should C^* ever abort the protocol, the simulator S^C will forward the abort to the trusted third party. Otherwise, it will follow $Hybrid3^{(C)}(k, x; r)$, controlling MOBILE and APPLICATION. If the cut-and-choose, input consistency check, or output proof of correctness fail, then S^C aborts to both C^* and the trusted third party. Otherwise, S^C outputs whatever C^* outputs. By Lemma 14-17, this simulator proves Theorem 1 when CLOUD is malicious.

Given the simulators S_M, S_A, and S_C, this proves the security of our protocol as stated in Theorem 1. □

3.6 Performance Analysis

We now characterize how garbled circuits perform in the constrained mobile environment with and without outsourcing. Two of the most important constraints for mobile devices are computation and bandwidth, and we show that order of magnitude improvements for both factors are possible with outsourced evaluation. We begin by describing our implementation framework and testbed before discussing results in detail.

3.6.1 Framework and Testbed

Our framework is based on the system designed by Kreuter et al. [103], hereafter referred to as *KSS* for brevity. We selected this system as our foundation and as our benchmarking comparison because it was the most efficient implementation of a two-party garbled circuit protocol at the time of our original publication. We contacted the authors of the Salus protocol [92] and requested the source code for their framework to compare the actual performance of their scheme with ours, but they were unable to release their code. Thus, an asymptotic comparison was the only fair comparison we could make to the only other existing outsourced scheme.

Using KSS as a foundation, we implemented our outsourced protocol and performed modifications to allow for the use of the mobile device in the computation. Notably, *KSS* uses the Message Passing Interface (MPI), for communication between the multiple nodes of the multi-core machines relied on for circuit evaluation. Our solution replaces MPI calls on the mobile device with sockets that communicate directly with the Generator and Proxy. To provide a consistent comparison, we revised the *KSS* codebase to allow for direct evaluation between the mobile device (the Evaluator) and the cloud-based Generator. The modifications required included removing the MPI library from the phone client and functions, which did not exist on the phone. The largest difficulty was changing the Intel specific instructions to generic instruction, which would work on the ARM processor of the mobile device.

We also informed the original authors of *KSS* of several problems we noticed. They in turn fixed the problems necessary for our trials. We found the following problems: generator's input consistency check was missing from one of the possible run environments, arrays did not work correctly in some cases, nested if statements did not work correctly, and for loops inside of if statements did not work correctly. We thank those authors for their assistance.

Our deployment platform consists of two Dell R610 servers, each containing dual 6-core Xeon processors with 32 GB of RAM and 300 GB 10K RPM hard drives, running the Linux 3.4 kernel and connected as a VLAN on an internal 1 Gbps switch. These machines perform

the roles of the Generator and Proxy, respectively, as described in Section 3.3.1. The mobile device acts as the Evaluator. We use a Samsung Galaxy Nexus phone with a 1.2 GHz dual-core ARM Cortex-A9 processor and 1 GB of RAM, running the Android 4.0 "Ice Cream Sandwich" operating system. We connect an Apple Airport Express wireless access point to the switch. The Galaxy Nexus communicates to the Airport Express over an 802.11n 54Mbps WiFi connection in an isolated environment to minimize co-channel interference. All tests are run 10 times with error bars on figures representing 95% confidence intervals.

We also ran tests using a 64-core server with 1 TB of memory where the phone was connected over a standard local network, which included additional traffic other than just our tests. This platform performed the roles of both the Generator and the Proxy. We experienced small but noticeable time differences depending upon what other tasks were being performed on the LAN.

When we use our 12-core servers both MOBILE and CLOUD each have their own 12-core server. In contrast, both parties are run on the same 64-core server. Each circuit can use two processes on our 64-core server or 1 process on each of our smaller 12-core servers. We ran our tests with the same security parameters as KSS for 80-bit security, although we vary the amount of circuits in our tests.

3.6.2 Execution Time

Our tests evaluated the following problems:

Millionaires: This problem models the comparison of two parties comparing their net worth to determine who has more money without disclosing the actual values. We perform the test on input values ranging in size from 4 to 8192 bits. In our circuit two comparisons are performed, one for MOBILE and one for MOBILE. There is one bit of output for each party.

Edit (Levenshtein) Distance: This is a string comparison algorithm that compares the number of modifications required to covert one string into another. We performed the comparison based on the circuit generated by Jha et al. [90] for strings sized between 4 and 128 bits. There are eight bits of output per party.

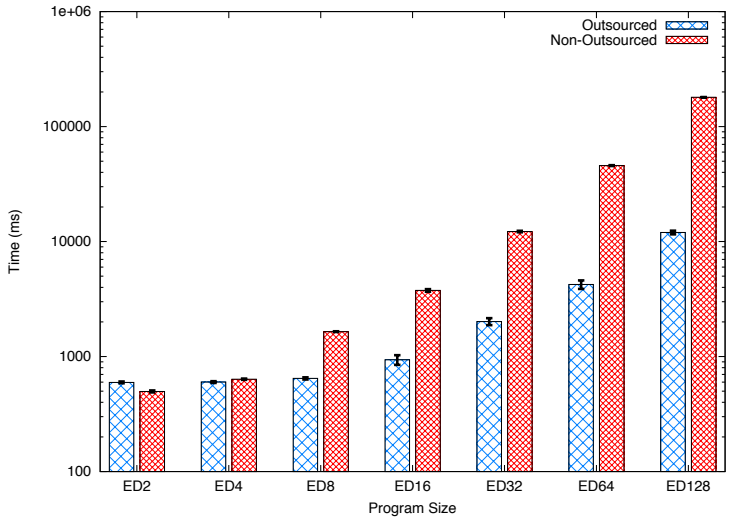


Figure 3: Execution time for the Edit Distance program of varying input sizes, with 2 circuits evaluated. On 12 core servers.

Set Intersection: This problem matches elements between the private sets of two parties without learning anything beyond the intersecting elements. We base our implementation on the SCS-WN protocol proposed by Huang et al. [81], and evaluate for sets of size 2 to 128. Each element in the set is 8 bits for 16 to 1024 bits of input per party. Each party receives $N * 8$ bits of output, the size of the largest possible result, where N is the size of the set.

AES: We compute an AES encrypt operation with a 128-bit key length, based on a circuit evaluated by Kreuter et al. [103]. One party enters in the key, 128 bits, and other party enters in text to encrypt, which is also 128 bits.

Since our protocol specifies all outputs must be under blinds, we also need to input output blinds, each of our circuits also inputs a blind for each party. The size of the blind corresponds to the size of the party's output.

Figure 3 shows the result of the edit distance computation for input sizes of 2 to 128 with two circuits evaluated. This comparison represents worst-case operation due to the cost of setup for a small number of small circuits - with input size 2, the circuit is only 122 total gates in size. For larger input sizes, however, outsourced computation becomes significantly faster. Note that the graph is logarithmic such that by the time strings of size 32 are evaluated, the outsourced execution is over 6 times faster than non-outsourced

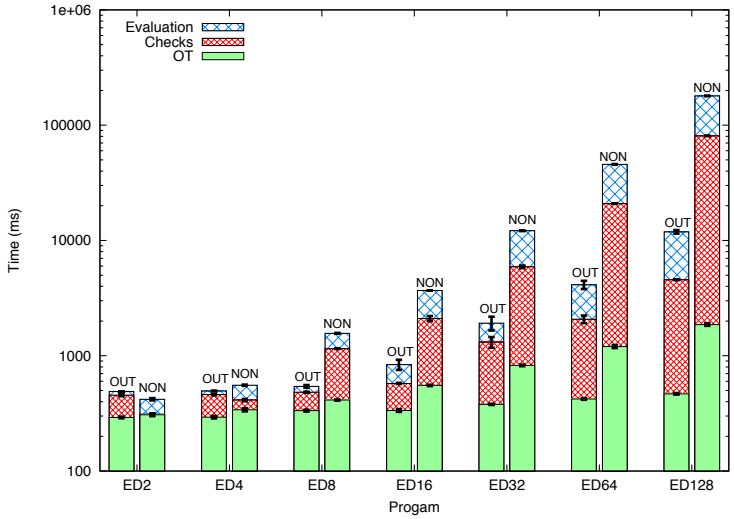


Figure 4: Execution time for significant stages of garbled circuit computation for outsourced and non-outsourced evaluation. The Edit Distance program is evaluated with variable input sizes for the two-circuit case. On 12 core servers.

execution, while for strings of size 128 (comprising over 3.4 million total gates), outsourced computation is over 16 times faster.

The reason for this becomes apparent when we examine Figure 4. There are three primary operations that occur during the SFE transaction: the oblivious transfer (OT) of participant inputs, the circuit commit (including the circuit consistency check), and the circuit evaluation. As shown in the figure, the OT phase takes 292 ms for input size 2, but takes 467 ms for input size 128. By contrast, in the non-outsourced execution, the OT phase takes 307 ms for input size 2, but increases to 1860 ms for input size 128. The overwhelming factor, however, is the circuit evaluation phase. It increases from 34 ms (input size 2) when the evaluation is complete by the time the checks finish on the phone to 7320 ms (input size 128) for the outsourced evaluation, a 215 factor increase. For non-outsourced execution however, this phase increases from 108 ms (input size 2) to 98800 ms (input size 128), a factor of 914 increase.

3.6.3 Evaluating Multiple Circuits

The security parameter for the garbled circuit check is $2^{-0.32\lambda}$ [103], where λ is the number of generated circuits. To ensure a sufficiently low probability (2^{-80}) of evaluating a corrupt

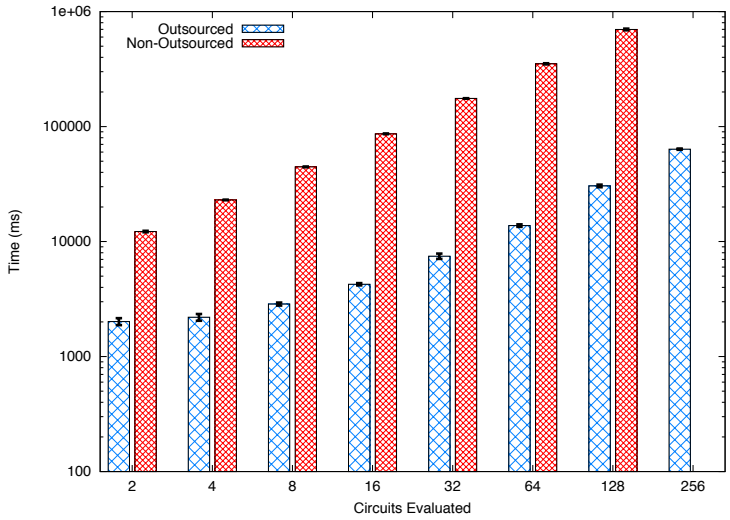


Figure 5: Execution time for the Edit Distance problem of size 32, with between 2 and 256 circuits evaluated. In the non-outsourced evaluation scheme, the mobile phone runs out of memory evaluating 256 circuits. On 12 core servers.

circuit, 256 circuits must be evaluated. However, there are increasing execution costs as increasing numbers of circuits are generated. Figure 5 shows the execution time of the Edit Distance problem of size 32 with between 2 and 256 circuits being evaluated. In the outsourced scheme, costs rise as the number of circuits evaluated increases. Linear regression analysis shows we can model execution time T as a function of the number of evaluated circuits λ with the equation $T = 243.2\lambda + 334.6$ ms, with a coefficient of determination R^2 of 0.9971. However, note that in the non-outsourced scheme, execution time increases over 10 times as quickly compared to outsourced evaluation. Regression analysis shows execution time $T = 5435.7\lambda + 961$ ms, with $R^2 = 0.9998$. Because in this latter case, the mobile device needs to perform all computation locally as well as transmit all circuit data to the remote parties, these costs increase rapidly. Figure 7 confirms this trend in all of our test programs, which clearly show that execution times for our protocol tend to be faster and increase at a slower rate than the execution times of KSS. Figure 6 provides more detail about each phase of execution. Note that the OT costs are similar between outsourced and non-outsourced execution for this circuit size, but that the costs of consistency checks and evaluation vastly increase execution time for non-outsourced execution.

Note as well that in the non-outsourced scheme, there are no reported values for 256

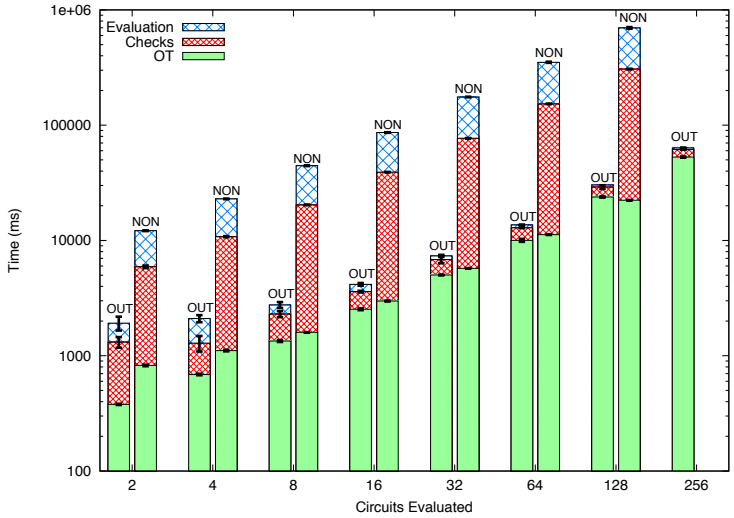


Figure 6: Microbenchmarks of execution time for Edit Distance with input size 32, evaluating from 2 to 256 circuits. Note that the y-axis is log-scale; consequently, the vast majority of execution time is in the check and evaluation phases for non-outsourced evaluation. On 12 core servers.

circuits, as the Galaxy Nexus phone ran out of memory before the execution completed. We observe that a single process on the phone is capable of allocating 512 MB of RAM before the phone would report an out of memory error, providing insight into how much intermediate state is required for non-outsourced evaluation. Thus, to handle circuits of any meaningful size with enough check circuits for a strong security parameter, the *only way* to be able to perform these operations is through outsourcing.

Our experiments span circuits from small to large input size, and from 8 circuits evaluated to the 256 circuits required for a 2^{-80} security parameter. Note that in many cases it is impossible to evaluate the non-outsourced computation because of the mobile device's inability to store sufficient amounts of state. Note as well that particularly with complex circuits such as set intersection, even when the non-outsourced evaluation is capable of returning an answer, it can require orders of magnitude more time than with outsourced evaluation. For example, evaluating the set intersection problem with 128 inputs over 32 circuits requires just over 55 seconds for outsourced evaluation but *over an hour and a half* with the non-outsourced *KSS* execution scheme. Outsourced evaluation represents a time savings of 98.92%. We also ran the non-outsourced *KSS* execution scheme on this server as

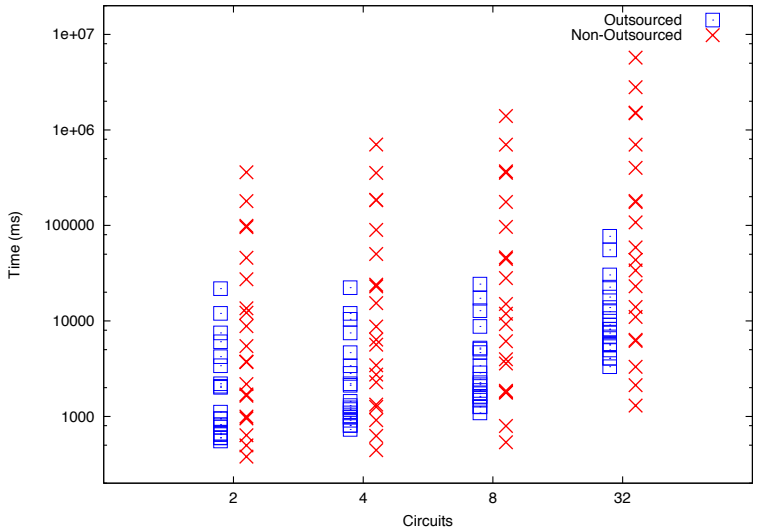


Figure 7: Execution time for all problems tested, with between 2 and 64 circuits evaluated. In the non-outsourced evaluation scheme, the mobile phone runs out of memory evaluating128 and 256 circuits. Observed that with the exception of a few programs with small input sizes, the execution times of the non-outsourced test programs cluster higher than the outsourced programs. This shows an overall execution time reduction achieved through outsourcing. On 12 core servers.

a comparison point.

Multicore Circuit Evaluation We briefly note the effects of multicore servers for circuit evaluation. The servers in our evaluation each contain dual 6-core CPUs, providing 12 total cores of computation. The computation process is largely CPU-bound: while circuits on the servers are being evaluated, each core was reporting approximately 100% utilization. This is evidenced by regression analysis when evaluating between 2 and 12 circuit copies; we find that execution time $T = 162.6\lambda + 1614.6$ ms, where λ is the number of circuits evaluated, with a coefficient of determination R^2 of 0.9903. As the number of circuits to be evaluated increases beyond the number of available cores, the incremental costs of adding new circuits becomes higher; in our observation of execution time for 12 to 256 circuits, our regression analysis provided the equation $T = 247.4\lambda - 410.6$ ms, with $R^2 = 0.998$. This demonstrates that evaluation of large numbers of circuits is optimal when every evaluated circuit can be provided with a dedicated core.

The results above show that as many-way servers are deployed in the cloud, it becomes

easier to provide optimal efficiency computing outsourced circuits. A 256-core machine would be able to evaluate 256 circuits in parallel to provide the accepted standard 2^{-80} security parameter. Depending on the computation performed, there can be a trade-off between a slightly weaker security parameter and maintaining *optimal* evaluation on servers with lower degrees of parallelism. In our testbed, optimal evaluation with 12 cores provides a security parameter of $2^{-3.84}$. Clearly more cores would provide stronger security while keeping execution times proportional to our results. A reasonable trade-off might be 32 circuits, as 32-core servers are readily available. Evaluating 32 circuits provides a security parameter of $2^{-10.2}$, equivalent to the adversary having less than a $\frac{1}{512}$ chance of causing the evaluator to compute over a majority of corrupt circuits. Stronger security guarantees on less parallel machines can be achieved at the cost of increasing execution time, as individual cores will not be dedicated to circuit evaluation. However, if a 256-core system is available, it will provide optimal results for achieving a 2^{-80} security parameter.

Different programs have different bottlenecks that affect the performance. When the bottleneck of the execution is on the mobile device, the speed of the computation cannot benefit by having more cores. Our analysis of the results revealed this occurs when a circuit has a high amount of inputs, relatively few gates in the circuit, and uses many cores working in parallel. In the aforementioned cases the gate execution completes before the mobile device completes the generator's input consistency. At 32 circuits the bottleneck is the input when we use our 12-core servers for AES, all millionaires programs, all set intersection programs other than size 128, edit distance 2,4, and 8. On our 64-core test server input is the bottleneck at 32 circuits for AES, all millionaires programs, all set intersection programs other than size 128, and edit distance 2,4,8, and 16.

In our tests we used the standard *KSS* implementation in our phone client for our non-outsourced tests. If we had used a more memory efficient implementation like PCF [104] and combined it with other memory optimizations we would have been able to evaluate all of our programs directly on a mobile phone. However, if this were the case our results would show further improvements in time and bandwidth our system saves over the mobile *KSS* client.

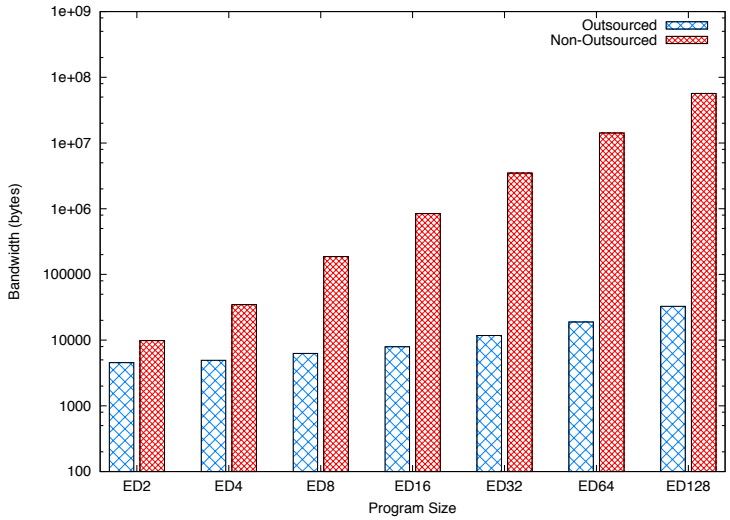


Figure 8: Bandwidth measurements from the phone to remote parties for the Edit Distance problem with varying input sizes, executing two circuits. On 12 core servers.

3.6.4 Bandwidth

For a mobile device, the costs of transmitting data are intrinsically linked to power consumption, as excess data transmission and reception reduces battery life. Bandwidth is thus a critical resource constraint. In addition, because of potentially uncertain communication channels, transmitting an excess of information can be a rate-limiting factor for circuit evaluation. Figure 8 shows the bandwidth measurement between the phone and remote parties for the edit distance problem with 2 circuits. When we compared execution time for this problem in Figure 3, we found that trivially small circuits could execute in less time without outsourcing. Note, however, that *there are no cases where the non-outsourced scheme consumes less bandwidth than with outsourcing.* Our other test programs showed similar results (see Figure 9), with the non-outsourced execution trending exclusively higher in bandwidth use than the outsourced protocol.

This is a result of the significant improvements garnered by using our outsourced oblivious transfer (OOT) construction described in Section 3.3. Recall that with the OOT protocol, the mobile device sends inputs for evaluation to the generator; however, after this occurs, all further evaluation until the final output verification from the cloud proxy occurs between the generator and the proxy, ensuring that further communication is not required

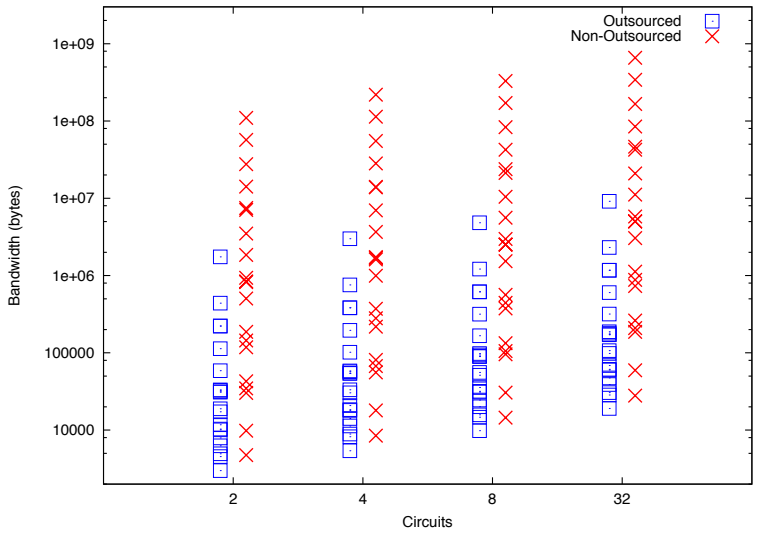


Figure 9: Bandwidth measurements from the phone to remote parties for all problems with varying input sizes, executing between 2 and 64 circuits. Note that the non-outsourced measurements cluster higher than the outsourced measurements, indicating that outsourcing the computation consistently saves bandwidth across the tested applications. On 12 core servers.

by the mobile device. Figure 8 shows that the amount of data transferred increases only nominally compared to the non-outsourced protocol. Apart from the initial set of inputs transmitted to the generator, data demands are largely constant. In particular, for large, complex circuits, the savings are vast: outsourced AES-128 requires 96.3% less bandwidth, while set intersection of size 128 requires 99.7% less bandwidth than in the non-outsourced evaluation. Remarkably, the edit distance 128 problem requires 99.95%, *over 1900 times less bandwidth*, for outsourced execution.

We performed an analytical analysis of what affects the amount of bandwidth. We determined input size is the only aspect of a program, which affects the amount of bandwidth used by the mobile device. The amount of gates in a circuit does not affect the amount of bandwidth used by the mobile device. MOBILE'S input (OTs) and APPLICATION'S input (consistency checks) both use bandwidth.

3.6.5 Network Latency

As network latency is a limiting factor on mobile phones, we wanted to see how our outsourced system performed in an environment with latency. We performed trials of our

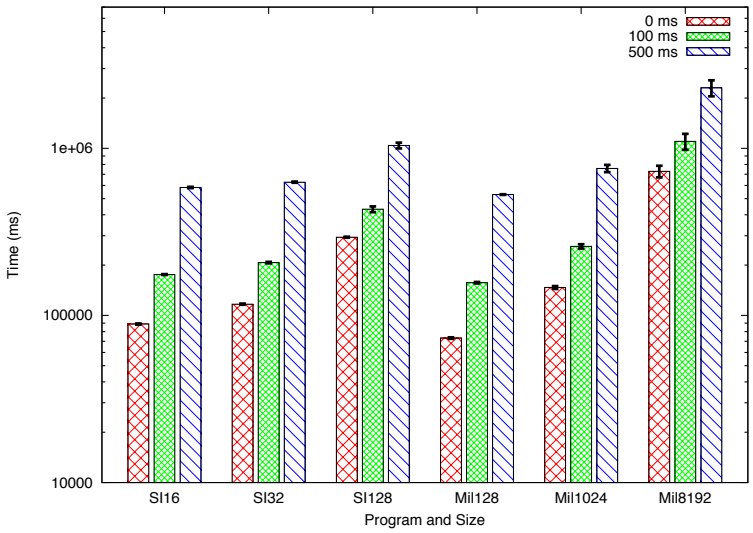


Figure 10: Execution time over varying network latency for the Edit Distance problem with varying input sizes, executing between 256 circuits. On 12 core servers.

execution system with two different amounts of latency added, 100ms and 500ms. We used Linux's *tc* command to emulate different latency amounts. It was found by Huang [78] that the median ping latency to a landserver on a 3G connection was between 180ms to 250ms.

In Figure 10 we present the results of our latency tests. The slowdown of added latency is not uniform across all of our tests. With the *Millionaires 8192*, we observed a slowdown of about 1.9X to 1.2X from 0 latency to 100ms latency, depending on the amount of circuits executed. Whereas *Set Intersection 16* had a slowdown of 2.8X to 2X when we added 100ms of latency. Making the transition from 0 latency to 500 ms makes the differences more apparent. We observed a slowdown of 3.7X to 2.8X for *Millionaires 8192*. Correspondingly, the *Set Intersection 16* had an observed slowdown between 9.7X to 6.4X.

The reason for the difference in the slowdown is due to the different bottlenecks different programs have in our system. For some programs the bottleneck will be at the phases necessary for input for the different parties, the oblivious transfer (large mobile input) and consistency check (large generator input). For other programs the bottleneck will be the garbled circuit generation and evaluation. A future goal is to improve the performance of our system in high latency environments.

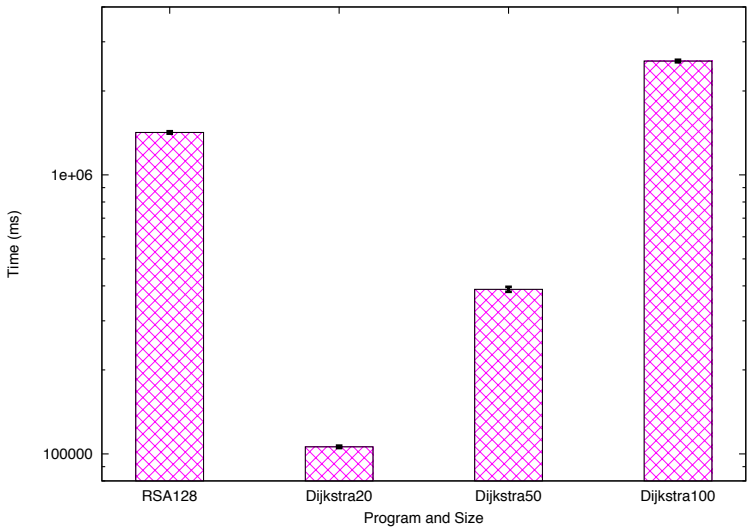


Figure 11: Execution time for the large circuit test applications with varying input sizes, executing 128 circuits. On 64 core servers.

Table 2: Bandwidth of 128-bit RSA and Dijkstra 20, 50, and 100. All entries are in Bytes.

	32 Circuits	64 Circuits	128 Circuits
RSA128	334629	672067	1346943
Dijkstra20	3862280	7770598	15587234
Dijkstra50	9575622	19266732	38648952
Dijkstra100	19087192	38405622	77042482

3.7 Evaluating Large Circuits

Beyond the standard benchmarks for comparing garbled circuit execution schemes, we aimed to provide compelling applications that exploit the mobile platform with large circuits that would be used in real-world scenarios. Based on our initial testing, these circuits are too large to evaluate directly on the mobile device using KSS due to the significant memory requirements. These experiments highlight the main practical contribution of our protocol, that we can now evaluate circuits that are *orders of magnitude* larger than was possible using previously existing techniques.

We discuss public-key cryptography and the Dijkstra shortest path algorithm, then describe how the latter can be used to implement a privacy-preserving navigation application for mobile phones.

3.7.1 Large Circuit Benchmarks

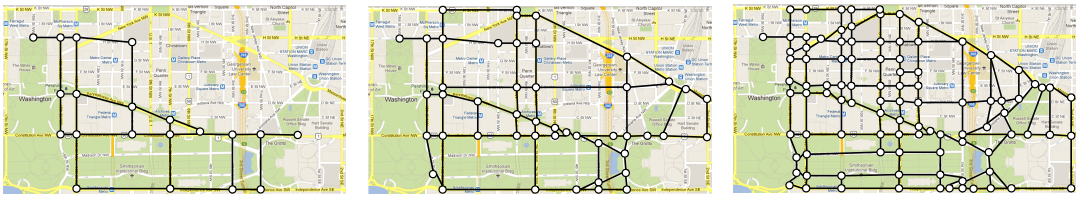
Figure 11 shows the execution time required for a blinded RSA circuit of input size of 128. For these tests we used our more powerful 64-core server. As described in Section 5.6, larger testbeds capable of executing 128 or 256 cores in parallel would be able to provide similar results for executing the 256 circuits necessary for a 2^{-80} security parameter as they could evaluate the added circuits in parallel. The main difference in execution time would come from the multiple OTs from the mobile device to the outsourced proxy. The RSA circuit has been previously evaluated with *KSS*, but never from the standpoint of a mobile device.

The RSA circuit inputs 128 bits each both parties as well the necessary output blinds. Both parties receive output 130 bits. The RSA circuit is the same circuit used in KSS12, which principally calculates a single modular exponentiation.

We only report the outsourced execution results, as the circuits are far too large to evaluate directly on the phone. As with the larger circuits described in Section 5.6, the phone runs out of memory from merely trying to store a representation of the circuit. Prior to optimization, the blinded RSA circuit is $192,537,834$ gates and afterward, comprises $116,083,727$ gates, or 774 MB in size.

Our Dijkstra's circuit assumes each node has a maximum degree of 4. Each edge weight is represented internally with 16 bits. Each node is internally represented with 8 bits. The program performs $N-1$ iterations of the algorithm in the circuit as it starts with the shortest path to the start node and adds the shortest path to a single node for every each iteration, where N is the number of nodes. The circuit inputs $N*96$ bits of input from Mobile, the map information, and 16 bits of input from Mobile, the starting and ending nodes. The circuit outputs $8*N$ bits to Mobile, the shortest path. The program and also inputs the necessary bits for the output blinds as well.

The implementation of Dijkstra's shortest-path algorithm results in very large circuits. The pre-optimized size of the shortest path circuit for 20 vertices is $20,288,444$ gates and after optimization is $1,653,542$ gates. The 100-node graph is even larger, with $168,422,382$ gates post optimization, 1124 MB in size. While it may be possible for existing protocols to evaluate circuits of similar size, it is significant that we are evaluating comparably massive



(a) 20 identified intersections. (b) 50 identified intersections. (c) 100 identified intersections.

Figure 12: Map of potential presidential motorcade routes through Washington, DC. As the circuit size increases, a larger area can be represented at a finer granularity.

circuits from a resource-constrained mobile device. Table 2 gives the amount of bandwidth these larger programs use.

3.7.2 Privacy-Preserving Navigation

Mapping and navigation are some of the most popular uses of a smartphone. Consider how directions may be given using a mobile device and an application such as Google Maps, without revealing the user's current location, their ultimate destination, or the route that they are following. That is, the navigation server should remain oblivious of these details to ensure their mutual privacy and to prevent giving away potentially sensitive details if the phone is compromised. Specifically, consider planning of the motorcade route for the recent Presidential inauguration. In this case, the route is generally known in advance but is potentially subject to change if sudden threats emerge. A field agent along the route wants to receive directions without providing the navigation service any additional details, and without sensitive information about the route loaded to the phone. Moreover, because the threats may be classified, the navigation service does not want the holder of the phone to be given this information directly.

To model this scenario, we overlay a graph topology on a map of downtown Washington D.C., encoding intersections as vertices. Edge weights are a function of their distance and heuristics such as potential risks along a graph edge. Figure 12 shows graphs generated based on vertices of 20, 50, and 100 nodes, respectively. Note that the 100-node graph (Figure 12c) encompasses a larger area and provides finer-grained resolution of individual intersections than the 20-node graph (Figure 12a).

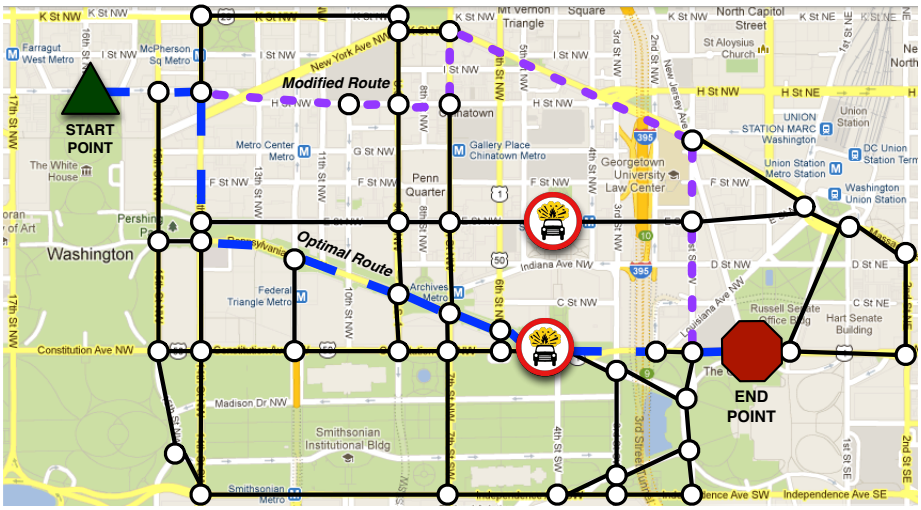


Figure 13: Motorcade route with hazards along the route. The dashed line represents the optimal route, while the dotted line represents the modified route that takes hazards into account.

There is a trade-off between detail and execution time, however; as shown in Figure 11, a 20-vertex graph can be evaluated in 1 minute 46 seconds, while a 100-vertex graph requires almost 43 minutes with 128 circuits in our 64-core server testbed. We anticipate that based on the role a particular agent might have on a route, they will be able to generate a route that covers their particular geographical jurisdiction and thus have an appropriately-sized route, with only certain users requiring the highest-resolution output. Additionally, as described in Section 3.6.3, servers with more parallel cores can simultaneously evaluate more circuits, giving faster results.

Figure 13 reflects two routes. The first, overlaid with a dashed line, is the shortest path under optimal conditions that is output by our directions service, based on origin and destination points close to the historical start and end points of the past six presidential inaugural motorcades. Now consider that incidents have happened along the route, shown in the figure as a car icon in a hazard zone inside a red circle. The agent recalculates the optimal route, which has been updated by the navigation service to assign severe penalties to those corresponding graph edges. The updated route returned by the navigation service is shown in the figure as a path with a dotted line. In the 50-vertex graph in Figure 12, the updated directions would be available in just over 135 seconds for 32-circuit evaluation,

and 196 and a half seconds for 64-circuit evaluation.

3.8 Conclusion

While garbled circuits offer a powerful tool for privacy-preserving computation, they typically assume participants with massive computing resources. Our first protocol solves this problem by outsourcing the most costly operations in garbled circuit evaluation from a resource-constrained mobile device to a cloud provider in the malicious setting. By extending existing garbled circuit evaluation techniques, this protocol significantly reduces both computational and network overhead on the mobile device while still maintaining the necessary checks for malicious or lazy behavior from all parties. Our outsourced oblivious transfer construction significantly reduces the communication load on the mobile device and can easily accommodate more efficient OT primitives as they are developed. The performance evaluation of this protocol shows dramatic decreases in required computation and bandwidth. For the edit distance problem of size 128 with 32 circuits, computation is reduced by 98.92% and bandwidth overhead reduced by 99.95% compared to non-outsourced execution. These savings are illustrated in our privacy-preserving navigation application, which allows a mobile device to efficiently evaluate a massive garbled circuit securely through outsourcing. These results demonstrate that the recent improvements in garbled circuit efficiency can be applied in practical privacy-preserving mobile applications on even the most resource-constrained devices.

CHAPTER IV

OUTSOURCING GARBLED CIRCUIT GENERATION

4.1 Introduction

Our first outsourcing protocol, which we described in Chapter 3 and refer to as the CMTB protocol, established the techniques necessary to outsource the evaluation of a garbled circuit. As our evaluation showed, this protocol for outsourcing allowed for significant efficiency savings on the mobile device, and turned previously infeasible computation problems into reality on the mobile platform. However, the CMTB protocol still required extended execution time for the larger test circuits due to several bottlenecks caused by the mobile device. Specifically, two phases of the protocol comprised an excessive portion of the execution: the OOT protocol and the claw-free input consistency checks. While the OT extensions by Ishai et al. allowed us to reduce the number of oblivious transfers required at the mobile device, these operations still require a significant amount of computation and act as a severe bottleneck on low-latency networks. Second, the group algebraic operations required for the claw-free consistency checks incur a significant execution time, and transmitting large group elements is a burden on the mobile bandwidth requirements.

In this chapter, we develop a new protocol for securely outsourcing garbled circuit generation rather than evaluation. We construct a protocol that offloads the role of generating the garbled circuit from the mobile device to the Cloud without exposing any private inputs or outputs. By building on the garbled circuit protocol of shelat and Shen [151] and choosing to outsource the circuit generation portion of the protocol, we eliminate a significant number of expensive public-key cryptography operations and rounds of communication used in both the OOT protocol and the input consistency check. The result is a more computationally and bandwidth efficient outsourcing protocol with stronger security guarantees.

Our Whitewash protocol and the following evaluation make the following contributions:

- **Develop a new outsourcing protocol:** We develop the Whitewash[1] outsourcing protocol, which allows a mobile device participating in a two-party secure function evaluation to outsource the generation of the garbled circuit. Our protocol assigns the mobile device the role of circuit generator instead of circuit evaluator, outsourcing a completely different set of operations from our previous outsourcing protocol and the malicious secure protocol of Kamara et al. [92]. By reversing the functions of the two players, we fully eliminate the requirement for any oblivious transfers, outsourced or otherwise, to or from the mobile device. This "simple" role reversal requires fundamentally redesigning the outsourcing techniques used in previous work, as well as new security proof formulations.

- **Formal verification and analysis:** We formally prove the security of our outsourcing techniques in the malicious model defined by Kamara et al. [92]. Unlike previous work, our protocol provides security when the mobile device is colluding with its Cloud provider against the application server. We then provide an analysis of the reduction in operations between our work and the outsourced oblivious transfer used in Chapter 3, as well as the Salus framework by Kamara et al. [92]. Specifically, our protocol requires more executions of a pseudorandom number generator in exchange for fewer algebraic group operations and zero-knowledge proofs. Moreover, we significantly reduce the number of rounds of communication required to the mobile device.

- **Implement and evaluate the performance of our protocol:** In our performance evaluation, we demonstrate a maximum improvement of 98% in execution time and 92% improvement in bandwidth overhead compared to our first protocol in Chapter 3 (with 75% and 60% average improvement, respectively). For a different test application, when compared to performing computation directly on the mobile device [103], we demonstrated a 96% and 90% improvement in execution time and bandwidth,

[1] A reference to Tom Sawyer, who "outsourced" his chores to his friends without ever revealing the true nature of the work.

respectively. These improvements allow for the largest circuits evaluated on any platform to be computed from a mobile device efficiently and with equivalent security parameters to non-mobile protocols.

This work was originally published at the Annual Computer Security Applications Conference [33].

4.2 Overview and Definitions

4.2.1 Protocol Goals and Summary

The primary reason for developing an outsourcing protocol for secure function evaluation is to allow two parties of asymmetrical computing ability to securely compute some result. Current two-party computation protocols assume both parties are equipped with equivalent computing resources and so require both parties to perform comparable operations. However, when a mobile device is taking part in computation with an application server, some technique is necessary to reduce the complexity of the operation on the mobile device. Ideally, we can make the mobile device perform some small number of operations that is independent of the size of the circuit being evaluated.

In constructing such a protocol, there are four goals that we would like to satisfy. The first of these goals is correctness. It is necessary that an outsourcing protocol must produce correct output even in the face of malicious players attempting to corrupt the computation. The second desirable guarantee is security. SFE protocols frequently use a simulation-based approach to defining and proving security, which we outline in detail below. Essentially, the goal is to show that each party can learn the output of the computed function and nothing else. Third, an ideal protocol would provide some guarantee of fair release. This guarantee ensures that either both parties receive their outputs from the computation, or neither party receives their output. Our protocol achieves this in all but one corruption scenario by treating the Cloud as an arbiter, who will simultaneously and fairly release the outputs of the protocol using one-time pads. In the scenario where the mobile device and Cloud are colluding, it is possible for the Cloud to terminate the protocol after the mobile device receives output but before the application server receives output. However,

this is inherently possible in most two-party garbled circuit protocols. The fourth goal of our protocol is efficiency. Our outsourcing protocol balances a minimal set of operations on the mobile device with efficient multiparty computation operations on the application server and Cloud.

Given these goals, we build our protocol in the two-party server-aided multiparty computation setting. This setting is composed of two parties, the mobile device and an application server, who wish to run a two-party secure computation while keeping both of their inputs private. To assist the mobile device, the server-aided setting adds a third party Cloud provider, which is independent and non-colluding with the application server (more on non-collusion in the following section). The Cloud performs cryptographic operations for the mobile device, but is not allowed to see any party's input or output from the computation.

To achieve our goals in this setting, we first select the most efficient two-party garbled circuit computation protocol to date that provides guarantees of correctness and security in the malicious model. We assign the mobile device the role of circuit generator in this protocol, and the application server is assigned the role of circuit evaluator. To outsource the circuit generation operations from the mobile device, we allow the device to generate short random seeds and pass these values to a Cloud computation provider, which then generates the garbled circuits using these seeds to generate randomness. Thus, the mobile device's work is essentially reduced to (1) generating random strings on the order of a statistical security parameter, and (2) garbling and sending its input values to the evaluating party. In this way, we develop a secure computation protocol where the mobile device performs work that is independent of the size of the function being evaluated.

4.2.2 Security Constructions

In the two-party computation protocol underlying our work, shelat and Shen implement a number of new and efficient cryptographic checks to ensure that none of the parties participating in the computation behave maliciously. We provide an overview of these security checks in the following section. We refer the reader to shelat and Shen's work for more formal definitions and proofs [151].

4.2.2.1 k-probe-resistant input encoding

When working with garbled circuit protocols in the malicious model, the generator has the ability to learn information about the evaluator's input by corrupting the wire labels sent during the oblivious transfer. This attack, known as selective failure, was first proposed by Mohassel and Franklin [123] as well as Kiraz and Schoenmakers [98]. In the server-aided setting, it is possible that the mobile device and the Cloud could collude and carry out this attack to recover the application server's input. To prevent this attack, shelat and Shen [151] implement an improved version of the k-probe-resistant input encoding mechanism originally proposed by Lindell and Pinkas [112]. In their protocol, the evaluator does not input her real input y to the computation, but chooses her input $\overline{y}$ such that $\mathbf{M} \cdot \overline{y} = y$ for a k-probe resistant matrix $\mathbf{M}$. Intuitively, the idea is that the generator would have to probe the evaluator's input approximately 2^k times before learning anything about her input y.

4.2.2.2 2-Universal Hash Function

A second concern with garbled circuits in the malicious model is that the generator may send different input values for each of the evaluated circuits from the cut-and-choose. As in the two-party setting, it is possible for the mobile device to submit inconsistent inputs to the application server in the server-aided setting. To ensure that the generator's inputs are consistent across evaluation circuits, shelat and Shen implement an efficient witness-indistinguishable proof, which computes a randomized, 2-universal hash of the input value using only arithmetic operations on matrices. Because of the regularity guarantees of a 2-universal hash, the outputs of these hash operations can be seen by the evaluator without revealing any information about the generator's inputs. However, if any of the hashed input values is inconsistent across evaluation circuits, the evaluator can infer that the generator provided inconsistent inputs, and can terminate the protocol.

4.2.2.3 Output proof of consistency

When a function being evaluated using garbled circuits has separate, private outputs for the generating and evaluating parties, it is necessary to ensure that the evaluating party does not tamper with the generating party's output. Since the output must be decoded from the garbled output wires for the majority check at the end of the protocol, if the output is only blinded with a one-time pad, this allows the evaluator the opportunity to change bits of the generator's output. Our setting faces the same potential for attack from the application server, who is responsible for evaluating the circuit and distributing the blinded output. Several techniques for preventing this kind of tampering have been proposed, but shelat and Shen's latest protocol [151] implements a witness-indistinguishable proof that uses only symmetric key cryptographic operations. After the evaluator sends the blinded output of computation to the generator, the proof guarantees to the generator that the output value he received was actually generated by one of the garbled circuits he generated. However, it keeps the index of the circuit that produced the output hidden, as this could leak information to the generator.

4.2.3 Security Model and non-collusion assumptions

Our definition of security is based on the definition proposed by Kamara et al. [92], which we specify for the two-party setting as in Chapter 3. As with our first protocol and other work in this area, we assume that the circuit evaluating server and circuit generating Cloud do not collude against the mobile device. However, while previous protocols restrict collusion between the Cloud and any party, the sub-linear work implication described in Chapter 3 only applies to cases when the Cloud is generating circuits and colludes with the evaluating party, or vice versa. In the Whitewash protocol, we prove security when the mobile device colludes with the Cloud against the evaluating web application. While this collusion scenario removes the fair release guarantee of our protocol, it in no way compromises the security guarantees of confidentiality of participant's inputs and outputs. Essentially, it reduces to the two-party computation scenario that the underlying protocol is proven to be secure in. Since the mobile device is paying the Cloud for computation services, we believe it is a

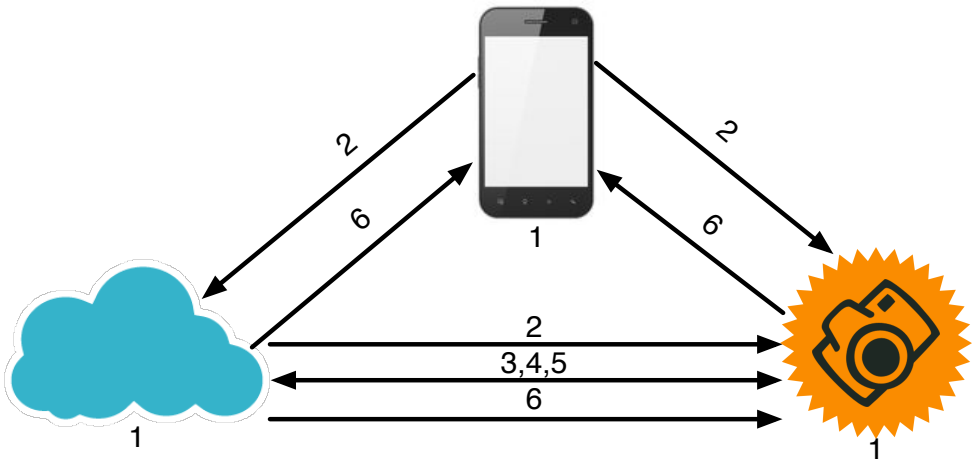


Figure 14: The complete Whitewash protocol. Note that MOBILE performs very little work compared to APPLICATION and CLOUD.

more realistic assumption to assume that a Cloud provider could collude maliciously with the paying customer, and note that our protocol is the first outsourcing protocol to provide any security guarantees in the face of collusion with the Cloud.

4.3 Protocol

4.3.1 Participants

Given a mobile device and a web or application server who wish to jointly compute a function, the three parties in this protocol correspond to the parties described in Section 3.3.1. However, we make an important distinction in the roles of the participants in the circuit garbling and evaluation. In the Whitewash protocol, MOBILE is tasked with garbling the circuit to be evaluated by APPLICATION. We show how to outsource these garbling operations to CLOUD.

4.3.2 Protocol

Common Inputs: Security parameters k (key length) and λ (the number of circuits generated for the cut-and-choose); a commitment scheme $com(x; c)$ with committed value x and commitment key c; and a function $f(x, y)$.

Private Inputs: MOBILE inputs x and APPLICATION inputs y.

Outputs: Two outputs f_a, f_m for APPLICATION and MOBILE, respectively.

Phase 1: Pre-computation

1. **Preparing inputs:** MOBILE randomly generates $r \in \{0,1\}^{2k+\log(k)}$ as his input to the 2-universal circuit. He also generates $e \in \{0,1\}^{|f_m|}$ as a one-time pad for his output. APPLICATION computes her k-probe-resistant matrix $\mathbf{M}$ and $\overline{y}$ such that $\mathbf{M} \cdot \overline{y} = y$. MOBILE'S input to the circuit will be $\overline{x} = x \| e \| r$ and APPLICATION'S input will be $\overline{y}$. We denote the set of indices $[m_a] = \{1, \cdots, |\overline{y}|\}$ and $[m_m] = \{1, \cdots, |\overline{x}|\}$.

2. **Preparing circuit randomness:** MOBILE generates random seeds $\{\rho^{(j)}\}_{j\in[\lambda]}$ for generating the circuits and sends them to CLOUD.

Phase 2: Input commitments

1. **Committing to Mobile's inputs:** For each circuit $j \in [\lambda]$, input bit $i \in [m_m]$, and $b \in \{0,1\}$ MOBILE uses $\rho^{(j)}$ to generate commitment keys $\theta_{i,b}^{(j)}$. Using the same random seeds, these keys will later be generated by CLOUD to commit to the input wire labels corresponding to MOBILE'S input. MOBILE then commits to his own inputs as $\{\Gamma^{(j)}\}_{j\in[\lambda]}$ as:

$$\Gamma^{(j)} = \{com(\theta_{i,\overline{x}_i}^{(j)}; \gamma_i^{(j)})\}_{i\in[m_m]}$$

   using independently generated random commitment keys $\gamma_i^{(j)}$. MOBILE sends $\{\Gamma^{(j)}\}_{j\in[\lambda]}$ to APPLICATION and the commitment keys $\{\gamma_i^{(j)}\}_{i\in[m_m],j\in[\lambda]}$ to CLOUD.

2. **Committing to Cloud's inputs:** To allow for a fair release of the outputs, CLOUD inputs one-time pads to blind both parties' outputs. CLOUD randomly generates $p_a \in \{0,1\}^{|f_a|}$ and $p_m \in \{0,1\}^{|f_m|}$, as well as $r_c \in \{0,1\}^{2k+\log(k)}$ as its input to the 2-universal circuit. We denote CLOUD'S input as $z = p_a \| p_m \| r_c$, and the indices of CLOUD'S input wires as $[m_c] = \{1, \cdots, |z|\}$.

   For each circuit $j \in [\lambda]$ and input bit $i \in [m_c]$, CLOUD uses $\{\rho^{(j)}\}_{j\in[\lambda]}$ to generates the garbled input wire keys $(K_{i,0}^{(j)}, K_{i,1}^{(j)}, \pi_i^{(j)})$, where $K_{i,0}^{(j)}, K_{i,1}^{(j)} \in \{0,1\}^k$ and the permutation bit $\pi_i^{(j)} \in \{0,1\}$. To locate the correct key for bit b on input wire w_i of circuit j, we designate the label $W_{i,b}^{(j)} = (K_{i,b}^{(j)}, b \oplus \pi_i^{(j)})$.

Let $\{w_{m_a+i}\}_{i\in[m_c]}$ be the input wires for CLOUD. CLOUD then commits to the label pairs for its input wires as $\{\Psi^{(j)}\}_{j\in[\lambda]}$, where

$$\Psi^{(j)} = \{com(W^{(j)}_{m_a+i,0\oplus\pi_i^{(j)}}; \psi^{(j)}_{i,0\oplus\pi_i^{(j)}}), com(W^{(j)}_{m_a+i,1\oplus\pi_i^{(j)}}; \psi^{(j)}_{i,1\oplus\pi_i^{(j)}})\}_{i\in[m_c]}$$

using commitment keys $\psi^{(j)}_{i,b}$ generated with the random seed $\rho^{(j)}$. CLOUD then commits to its inputs as $\{\Xi^{(j)}\}_{j\in[\lambda]}$ as:

$$\Xi^{(j)} = \{com(\psi^{(j)}_{i,z_i}; \xi^{(j)}_i)\}_{i\in[m_c]}$$

using independently generated random commitment keys. CLOUD sends $\{\Psi^{(j)}\}_{j\in[\lambda]}$ and $\{\Xi^{(j)}\}_{j\in[\lambda]}$ to APPLICATION.

Phase 3: Circuit construction

1. **Constructing the objective circuit:** APPLICATION sends $\mathbf{M}$ to CLOUD, then APPLICATION and CLOUD run a coin flipping protocol to randomly determine the 2-universal hash matrix $\mathbf{H} \in \{0,1\}^{k\times m_m}$. These two matrices can be used to generate the new circuit C that computes the function $g : (\overline{x}, \overline{y}) \to (\perp, (h_m, h_c, c_a, c_m))$, where $h_m = \mathbf{H}\cdot\overline{x}, h_c = \mathbf{H}\cdot z, g_m = f_a(x, \mathbf{M}\cdot\overline{y}), c_m = g_m \oplus e \oplus p_m, g_a = f_s(x, \mathbf{M}\cdot\overline{y})$, and $c_a = g_a \oplus p_a$. MOBILE will need the values $h_c\|c_m$ to recover his output. We denote the set of indices corresponding to these values as $O_m = \{1, \cdots, |h_c| + |c_m|\}$.

2. **Committing to input and output wire label pairs:** Using the same method as in Phase 2, CLOUD uses $\{\rho^{(j)}\}_{j\in[\lambda]}$ to generate the input wire keys for both APPLICATION and MOBILE'S input as well as the output wire keys for MOBILE'S output (these output keys must be committed for the witness indistinguishable proof of MOBILE'S output correctness). Let $\{w_i\}_{i\in[m_m]}$ be the input wires for MOBILE, $\{w_{m_m+i}\}_{i\in[m_a]}$ be the input wires for APPLICATION, and $\{w_i\}_{i\in O_m}$. CLOUD then commits to the label pairs in MOBILE'S input, APPLICATION'S input, and MOBILE'S

output as $\{\Theta^{(j)}, \Omega^{(j)}, \Phi^{(j)}\}_{j\in[\lambda]}$, where

$$\Theta^{(j)} = \{com(W^{(j)}_{i,0\oplus\pi_i^{(j)}}; \theta^{(j)}_{i,0\oplus\pi_i^{(j)}}), com(W^{(j)}_{i,1\oplus\pi_i^{(j)}}; \theta^{(j)}_{i,1\oplus\pi_i^{(j)}})\}_{i\in[m_m]}$$
$$\Omega^{(j)} = \{com(W^{(j)}_{m_m+i,0}; \omega_i^{(j)}), com(W^{(j)}_{m_m+i,1}; \omega_i^{(j)})\}_{i\in[m_a]}$$
$$\Phi^{(j)} = \{com(W^{(j)}_{i,0}; \phi_i^{(j)}), com(W^{(j)}_{i,1}; \phi_i^{(j)})\}_{i\in O_m}$$

using commitment keys generated with the random seed $\rho^{(j)}$. Cloud then sends these commitments to Application.

Phase 4: Oblivious transfers (OT)

1. **Oblivious transfers:** Cloud and Application execute m_a input oblivious transfers and λ circuit oblivious transfers as follows:

   (a) **Input:** For each $i \in [m_a]$, both parties run a $\binom{2}{1}$-OT where Cloud inputs

   $$\left(\{(W^{(j)}_{m_m+i,0}; \omega_i^{(j)})\}_{j\in[\lambda]}, \{(W^{(j)}_{m_m+i,1}; \omega_i^{(j)})\}_{j\in[\lambda]}\right)$$

   while Application inputs $\overline{y}_i$. Once Application receives all of her garbled input wire labels, she uses the decommitment keys obtained in the OTs to check the committed wire values in $\{\Omega^{(j)}\}_{j\in[\lambda]}$. If any of the labels received in the OT do not match the committed wire labels, Application terminates the protocol.

   (b) **Circuit:** Application selects a set of circuits to be evaluated $S \subset [\lambda]$ such that $|S| = \frac{2\lambda}{5}$, as in shelat and Shen's protocol [150]. She represents this set with a bit string $s \in \{0,1\}^\lambda$ such that the j^{th} bit $s_j = 1$ if $j \in S$ and $s_j = 0$ otherwise. Application and Cloud perform λ $\binom{2}{1}$-OTs where, for every $j \in [\lambda]$, Cloud inputs $(\rho^{(j)}, (\{\gamma_i^{(j)}\}_{i\in[m_m]} \| \{\Xi_i^{(j)}\}_{i\in[m_c]}))$, while Application inputs s_j. This allows Application to learn either the randomness used to generate the check circuits or Mobile and Cloud's inputs for the evaluation circuits without Cloud knowing which circuits are being checked or evaluated.

Phase 5: Evaluation

1. **Circuit evaluation:** Using $\rho^{(j)}$, Cloud garbles the objective circuit C as $G(C)^{(j)}$ for all $j \in [\lambda]$ and pipelines these circuits to Application using Huang's technique [82]. Depending on whether the circuit is a check circuit or an evaluation circuit, Application performs one of two actions:

   (a) **Check:** For each $j \in [\lambda] \backslash S$, Application checks to see if $\rho^{(j)}$ can correctly regenerate the committed wire values $\{\Theta^{(j)}, \Omega^{(j)}, \Phi^{(j)}, \Psi^{(j)}\}$ and the circuit $G(C)^{(j)}$.

   (b) **Evaluate:** For each $j \in S$, Application checks that she can correctly decommit Mobile's input by recovering half of $\Theta^{(j)}$ from the keys committed in $\Gamma^{(j)}$. She does the same for Cloud's input, recovering half of $\Psi^{(j)}$ from the keys committed in $\Xi^{(j)}$

   If any of the above checks fail, Application aborts the protocol. Otherwise, she evaluates the circuits $\{G(C)^{(j)}\}_{j \in [\lambda] \backslash S}$. Each circuit outputs the values $(h_m^{(j)}, h_c^{(j)}, c_a^{(j)}, c_m^{(j)})$ for $j \in [\lambda] \backslash S$.

2. **Majority output selection and consistency check:** Let (h_m, h_c, c_a, c_m) be the output of the majority of the evaluated circuits. If no majority value exists, Application aborts the protocol. Otherwise, she checks that $h_m^{(j)} = h_m$ and $h_c^{(j)} = h_c$ for all $j \in [\lambda] \backslash S$. If any of Mobile or Cloud's hashed input values do not match, Application aborts the protocol.

Phase 6: Output proof and release

1. **Proof of output authenticity:** Application and Mobile perform the proof of output authenticity from shelat and Shen's protocol [151] using the commitments to Mobile's output wires $\{\Phi^{(j)}\}_{j \in [\lambda] \backslash S}$ and the values $h_c \| c_m$.

2. **Output release:** Cloud simultaneously releases the input one-time pads p_a and p_m to Application and Mobile. Application and Mobile then hash the pads and check to see if the hash values output by the circuit $h_c = H \cdot p_a \| p_m$. If the hashes do not match, Application and Mobile abort the protocol. Otherwise, Application receives $c_a \oplus p_a$ as her output and Mobile receives $c_m \oplus p_m \oplus e$ as his output.

4.4 Proof of Security

Following the security definition from Section 5.2, we prove the following theorem.

Theorem 2. *The Whitewash outsourced two-party SFE protocol securely computes a function $f(x, y)$ in the following three corruption scenarios: (1)* Cloud *is malicious and non-cooperative with respect to the rest of the parties, while all other parties are semi-honest; (2) All but one party providing input is malicious, while* Cloud *is semi-honest; or (3)* Cloud *and* Mobile *are malicious and colluding, while* Application *is semi-honest.*

Note that previous outsourcing schemes [92] are only secure in corruption scenarios (1) and (2).

4.4.1 Malicious Application A^*

Consider when Application can perform arbitrarily malicious actions while Mobile and Cloud follow the protocol in a semi-honest manner. We note that the operations performed by A^* and the messages received by A^* are nearly identical to the malicious evaluator P_2^* from shelat and Shen's proof of their two-party computation scheme [151]. We note here four slight alterations necessary to their simulator S_2, none of which change their proof of security. We call the modified simulator S_A.

1. **Input generation:** When S_A generates a random input $\overline{x}'$ for Mobile, it also generates a random input $\overline{z}'$ for Cloud. Because this input is chosen from a uniform distribution in both the real and the ideal world, it is statistically indistinguishable.

2. **Input commitments:** When S_A generates the input commitments $\{\Gamma^{(j)}\}_{j\in\lambda}$, it also generates commitments $\{\Xi^{(j)}\}_{j\in\lambda}$ to commit to Cloud's input.

3. **Wire label commitments:** When S_A generates the commitments to its input wires $\{\Theta^{(j)}\}_{j\in\lambda}$, it also generates commitments to Cloud's input wire labels $\{\Psi^{(j)}\}_{j\in\lambda}$.

4. **Output proof:** If A^* successfully proves the correctness of Mobile's output, the simulator S_A delivers the random input $\overline{z}'$ to A^*. As stated above, this input is statistically indistinguishable from Cloud's input in the real world.

Given the existence of the simulator S_A, this proves security when the evaluating party A^* is malicious (scenario 2).

4.4.2 Malicious Mobile M^*

Consider when MOBILE can perform arbitrarily malicious actions while APPLICATION and CLOUD follow the protocol in a semi-honest manner. We construct a simulator S_M in the ideal world to simulate MOBILE'S view of a real execution of the protocol. Note that the simulator does not have the other parties' inputs, nor does it know what input the malicious M^* will use. Thus, MOBILE'S inputs and commitments must be checked, and the output proof and result of computation must be simulated. Consider the following hybrid of experiments.

$\boldsymbol{Hybrid1^{(M)}(k,x;r)}$: This experiment is identical to the experiment $REAL^{(M)}(k,x;r)$ except that the experiment receives the values $\{\rho^{(j)}\}_{j\in\lambda}, \{\gamma^{(j)}\}_{j\in\lambda}$, and $\{\Gamma^{(j)}\}_{j\in\lambda}$ from M^* and uses them to recover M^*s input. If for any $j \in S$, the decommitment $\Gamma^{(j)}$ cannot reveal M^*s input $\overline{x}^{*(j)}$, the simulator aborts.

Lemma 18. $REAL^{(M)}(k,x;r) \overset{c}{\approx} Hybrid1^{(M)}(k,x;r)$

Proof. Because the experiment is in control of APPLICATION and CLOUD, for any $j \in \lambda$ we know that the commitment $\Theta^{(j)}$ is constructed correctly using $\rho^{(j)}$. Thus, the only possible way that the experiment will not uncover the value for some $\overline{x}^{*(j)}$ is if $\{\theta^{(j)}_{i,\overline{x}^*_i}\}_{i\in[m_m]}$, when decommitted from $\Gamma^{(j)}$ using $\gamma^{(j)}$ correctly decommits the $i \oplus 1$ half of $\Theta^{(j)}$, which happens with negligible probability based on the binding property of the commitment. Otherwise, at least one of the two commitments $\Gamma^{(j)}$ or $\Theta^{(j)}_i$ must fail to decommit, in which case both experiments abort. □

$\boldsymbol{Hybrid2^{(M)}(k,x;r)}$: This experiment is identical to the experiment $Hybrid1^{(M)}(k,x;r)$ except that if the extracted inputs are inconsistent, the experiment aborts.

Lemma 19. $Hybrid1^{(M)}(k,x;r) \overset{c}{\approx} Hybrid2^{(M)}(k,x;r)$

Proof. This follows from the 2-universal hash check of consistency. Since all of the circuits are generated by the experiment as CLOUD, they are all constructed correctly. Following

from Lemma G.10 in shelat and Shen's proof [151], indistinguishability holds here. □

***Hybrid3*$^{(M)}(k, x; r)$**: This experiment is identical to the experiment $Hybrid2^{(M)}(k, x; r)$ except that the experiment passes x^* to the trusted third party and receives $f_m(x^*, y)$ in return. It then randomly selects an evaluated circuit $G(C)^{(j)}$ and uses the output keys from that circuit to run the output proof of correctness for $f_m(x^*, y) \oplus e^* \oplus p_m$ with M^*.

Lemma 20. $Hybrid2^{(M)}(k, x; r) \overset{c}{\approx} Hybrid3^{(M)}(k, x; r)$

Proof. This follows from the witness-indistinguishability property of the output proof, which guarantees that the index of the circuit output being sent remains hidden. Indistinguishability follows directly from Lemma G.12 in shelat and Shen's proof [151]. □

***Hybrid4*$^{(M)}(k, x; r)$**: This experiment is identical to the experiment $Hybrid3^{(M)}(k, x; r)$ except that the experiment selects random inputs for Application a' and Cloud z' following the parameters of the protocol.

Lemma 21. $Hybrid3^{(M)}(k, x; r) \overset{c}{\approx} Hybrid4^{(M)}(k, x; r)$

Proof. This follows from the security guarantees of the garbled circuit itself. Since the output of the circuit in the real world matches the output of the trusted third party in the ideal world, M^* learns nothing from the output received. Since the output produced by replacing Application's input with random inputs is never returned to M^*, he cannot distinguish between Application's real inputs and the random inputs. Finally, since Cloud's input is two pseudorandom strings in the real protocol, it is statistically indistinguishable from the experiment's choice of z'. Thus, indistinguishability holds even when z' is revealed in the output release phase. □

Lemma 22. $Hybrid4^{(M)}(k, x; r)$ *runs in polynomial time.*

Proof. This follows trivially from the fact that the main protocol runs in polynomial time. Since the experiment does not perform any additional actions beyond the main protocol, it also runs in polynomial time. □

$Hybrid4^{(M)}(k, x; r)$ is identical to the simulator S_M running in the ideal world. The simulator runs M^* and controls Application and Cloud. If any of the consistency checks fails, S_M terminates the protocol. Otherwise, it delivers M^*s input to the trusted third party in $Hybrid3^{(M)}(k, x; r)$, and outputs whatever M^* outputs. By Lemma 18-22, this simulator proves Theorem 2 when the mobile device is malicious (scenario 2).

4.4.3 Malicious Cloud C^*

Consider when Cloud can perform arbitrary malicious actions while Application and Mobile follow the protocol in a semi-honest manner. We construct a simulator S_C in the ideal world to simulate Cloud's view of a real execution of the protocol. Note that since the simulator does not have the other parties' inputs, nor does it know what input the malicious C^* will use. Thus, the inputs, commitments, and circuits generated by the Cloud must be checked, and the oblivious transfers must be simulated. Consider the following hybrid of experiments.

$\boldsymbol{Hybrid1^{(C)}(k, x; r)}$: This experiment is identical to the experiment $REAL^{(C)}(k, x; r)$ except that instead of running the circuit oblivious transfers, the experiment invokes the simulator S_{OT}, which recovers both of C^*s inputs to the oblivious transfer (i.e., the random coins $\{\rho^{(j)}\}_{j\in\lambda}$ and the commitment keys $\{\xi^{(j)}\}_{j\in\lambda}$).

Lemma 23. $REAL^{(C)}(k, x; r) \overset{c}{\approx} Hybrid1^{(C)}(k, x; r)$

Proof. Based on the malicious security of the oblivious transfer primitive, we know that S_{OT} exists. The proof of this lemma follows directly from Lemma G.7 in shelat and Shen's security proof [151]. □

$\boldsymbol{Hybrid2^{(C)}(k, x; r)}$: This experiment is identical to the experiment $Hybrid1^{(C)}(k, x; r)$ except that if more than $\lambda/5$ circuits are incorrectly constructed, then the experiment aborts.

Lemma 24. $Hybrid1^{(C)}(k, x; r) \overset{c}{\approx} Hybrid2^{(C)}(k, x; r)$

Proof. For a circuit to be incorrectly constructed means that the commitments $\{\Theta^{(j)}, \Omega^{(j)}, \Phi^{(j)}, \Psi^{(j)}\}$ and the circuit $G(C)^{(j)}$, for $j \in \lambda$, cannot be reconstructed given the objective circuit C and the randomness $\rho^{(j)}$. Again, this lemma follows directly from Lemma G.8 in shelat and Shen's proof [151]. □

***Hybrid3*$^{(C)}(k, x; r)$**: This experiment is identical to the experiment $Hybrid2^{(C)}(k, x; r)$ except that the experiment will abort if C^*s private inputs cannot be recovered for at least $\lambda/5$ of the evaluation circuits.

Lemma 25. $Hybrid2^{(C)}(k, x; r) \overset{c}{\approx} Hybrid3^{(C)}(k, x; r)$

Proof. From the previous lemma, we know that $4\lambda/5$ of the circuits are correctly constructed. This implies that of the $2\lambda/5$ circuits chosen to be evaluated, at least $\lambda/5$ are "good" circuits. Let these "good" circuits be denoted as G, where $|G| \geq \lambda/5$. Assume for contradiction that there is some $j \in G$ where $\overline{z}^{*(j)}$ cannot be recovered. The only possible way that the experiment will not uncover the value for some $\overline{z}^{*(j)}$ is if $\{\psi^{(j)}_{i,\overline{z}^*_i}\}_{i\in[m_c]}$, when decommitted from $\Xi^{(j)}$ using $\xi^{(j)}$ correctly decommits the $i \oplus 1$ half of $\Psi^{(j)}$, which happens with negligible probability based on the binding property of the commitment. Otherwise, at least one of the two commitments $\Xi^{(j)}$ or $\Psi^{(j)}_i$ must fail to decommit, in which case both experiments abort.

□

***Hybrid4*$^{(C)}(k, x; r)$**: This experiment is identical to the experiment $Hybrid3^{(C)}(k, x; r)$ except that the experiment aborts if any of C^*s inputs to the good circuits in G is inconsistent.

Lemma 26. $Hybrid3^{(C)}(k, x; r) \overset{c}{\approx} Hybrid4^{(C)}(k, x; r)$

Proof. Informally, this proof follows from the 2-universal hash check used in the circuit. This lemma follows directly from Lemma G.10 in shelat and Shen's proof [151]. □

***Hybrid5*$^{(C)}(k, x; r)$**: This experiment is identical to the experiment $Hybrid4^{(C)}(k, x; r)$ except that the experiment chooses a random input for APPLICATION y' and computes $\overline{y}'$ such that $\mathbf{M} \cdot \overline{y}' = y'$ and uses that as input to the input oblivious transfers.

Lemma 27. $Hybrid4^{(C)}(k, x; r) \stackrel{c}{\approx} Hybrid5^{(C)}(k, x; r)$

Proof. Informally, this proof follows from the choose security of the OT primitive. This lemma follows directly from Lemma G.14 in shelat and Shen's proof [151]. □

***Hybrid6*$^{(C)}(k, x; r)$**: This experiment is identical to the experiment $Hybrid5^{(C)}(k, x; r)$ except that the experiment chooses a random input for MOBILE x' and computes $\overline{x}'$ by concatenating random strings e' and r' to x' and uses these inputs as input to the computation.

Lemma 28. $Hybrid5^{(C)}(k, x; r) \stackrel{c}{\approx} Hybrid6^{(C)}(k, x; r)$

Proof. Because C^* never receives MOBILE'S inputs in any form, or output from the computation, he cannot distinguish between using MOBILE'S real inputs and random inputs chosen by the experiment. Since C^* only ever sees MOBILE'S input commitment keys $\{\gamma^{(j)}\}_{j\in\lambda}$, which are pseudorandom strings in both experiments, these strings are statistically indistinguishable as well. □

***Hybrid7*$^{(C)}(k, x; r)$**: This experiment is identical to the experiment $Hybrid6^{(C)}(k, x; r)$ except that the simulator runs the function $f(x', y')$ for the inputs randomly chosen in the previous lemma. If $f_m(x', y') \oplus e^* \oplus p_m$ and $f_a(x', y') \oplus p_a$ do not match a majority of the evaluation outputs, the experiment aborts.

Lemma 29. $Hybrid6^{(C)}(k, x; r) \stackrel{c}{\approx} Hybrid7^{(C)}(k, x; r)$

Proof. This follows trivially from the fact that at least $\lambda/5$ circuits are correctly constructed and that C^*s inputs to those circuits are consistent. Thus, the majority output will be exactly $f_m(x', y') \oplus e^* \oplus p_m$ and $f_a(x', y') \oplus p_a$. □

***Hybrid8*$^{(C)}(k, x; r)$**: This experiment is identical to the experiment $Hybrid7^{(C)}(k, x; r)$ except that when C^* returns the one-time pads w^* in the output release, the experiment aborts if $w^* \neq z^*$, where z^* is the consistent input to the good circuits in G. Otherwise, the experiment sends z^* to the trusted third party as C^*s input.

Lemma 30. $Hybrid7^{(C)}(k, x; r) \stackrel{c}{\approx} Hybrid8^{(C)}(k, x; r)$

Proof. This follows from the collision-resistance of the 2-universal hash family. C^* can only return a value w^* such that $\mathbf{H} \cdot w^* = \mathbf{H} \cdot z^*$ with negligible probability, and so the experiments are indistinguishable. □

Lemma 31. *$Hybrid8^{(C)}(k, x; r)$ runs in polynomial time.*

Proof. This follows trivially from the fact that the main protocol runs in polynomial time. Since the experiment only evaluates $f(\cdot, \cdot)$ (which is also polynomial time) in addition to the main protocol, it also runs in polynomial time. □

$Hybrid8^{(C)}(k, x; r)$ is identical to the simulator S_C running in the ideal world. The simulator runs C^* and controls Application and Mobile. If any of the consistency checks fails, S_C terminates the protocol. Otherwise, it delivers C^*s input to the trusted third party when it is completes $Hybrid8^{(C)}(k, x; r)$, and outputs whatever C^* outputs. By Lemma 23-31, this simulator proves Theorem 2 when Cloud is malicious (scenario 1).

4.4.4 Malicious and colluding Mobile and Cloud MC^*

Consider when Mobile and Cloud can perform arbitrary malicious actions and share arbitrary information while Application follows the protocol in a semi-honest manner. We observe that this scenario is equivalent to a malicious generator P_1^* in shelat and Shen's proof of security [151], with some modifications to the lemmas to account for communicating with two parties and to account for Cloud's added input. We also note that in this scenario, the malicious and colluding MC^* may terminate the protocol early, preventing Application from receiving her output. However, this is possible on the evaluator's side in shelat and Shen's protocol, so we consider fair release a separate guarantee from security. We describe the changes to each hybrid experiment in shelat and Shen's proof below, as well as noting slight changes to the proofs of each lemma.

$\boldsymbol{Hybrid1^{(MC)}(k, x; r)}$: This experiment is identical to the experiment $REAL^{(MC)}(k, x; r)$ except that instead of running the circuit oblivious transfers, the experiment invokes the simulator S_{OT}, which recovers both of C^*s inputs to the oblivious transfer (i.e., the random

coins $\{\rho^{(j)}\}_{j\in\lambda}$ and the commitment keys $\{\gamma^{(j)}\}_{j\in\lambda}, \{\xi^{(j)}\}_{j\in\lambda}$).

The proof of this hybrid follows directly from shelat and Shen, only it is extended to recover the commitments to C^*s input as well as M^*s.

$\boldsymbol{Hybrid2^{(MC)}(k,x;r)}$: This experiment is identical to the experiment $Hybrid1^{(MC)}(k,x;r)$ except that if more than $\lambda/5$ circuits are incorrectly constructed, then the experiment aborts.

Again, this follows directly from shelat and Shen. However, the commitments to the CLOUD's input wires $\{\Psi^{(j)}\}_{j\in\lambda}$ must also be checked.

$\boldsymbol{Hybrid3^{(MC)}(k,x;r)}$: This experiment is identical to the experiment $Hybrid2^{(MC)}(k,x;r)$ except that the experiment will abort if both M^* and C^*s private inputs cannot be recovered for at least $\lambda/5$ of the evaluation circuits.

Lemma 32. $Hybrid2^{(MC)}(k,x;r) \overset{c}{\approx} Hybrid3^{(MC)}(k,x;r)$

Proof. This lemma holds in the same manner as Lemma 25 when only the CLOUD is malicious. Since the commitments $\Theta^{(j)}$ and $\Psi^{(j)}$ are constructed correctly, the only way that the input of either M^* or C^* cannot be recovered is if the decommited values from $\Gamma^{(j)}$ and $\Xi^{(j)}$ decommitted the wrong halves of the commitments $\Theta^{(j)}$ and $\Psi^{(j)}$ respectively. This would imply that M^* or C^* was able to break the binding property of the commitment, which can only happen with negligible probability. □

$\boldsymbol{Hybrid4^{(MC)}(k,x;r)}$: This experiment is identical to the experiment $Hybrid3^{(MC)}(k,x;r)$ except that the experiment aborts if any of M^* or C^*s inputs to the good circuits in G are inconsistent.

Again, this follows directly from shelat and Shen, expanded to handle the inputs of both malicious parties.

$\boldsymbol{Hybrid5^{(MC)}(k,x;r)}$: This experiment is identical to the experiment $Hybrid4^{(MC)}(k,x;r)$ except that the input recovered in the previous hybrid, $\overline{x}^* = x^*||r^*||e^*$ and $\overline{z}^* = p_a^*||p_m^*$ are forwarded to the trusted third party, which returns $f(x^*, y, z^*)$. The experiment aborts if the majority output of the computation does not match $f_m(x^*, y) \oplus e^* \oplus p_m$.

Here we extend shelat and Shen to include the input from CLOUD, which is added in as a blind to the output of computation in the real protocol.

$\boldsymbol{Hybrid6^{(MC)}(k,x;r)}, \boldsymbol{Hybrid7^{(MC)}(k,x;r)}, \boldsymbol{Hybrid8^{(MC)}(k,x;r)}$: These hybrid experiments are identical to the hybrids in shelat and Shen's proof. So, we invoke them directly and the proofs follow as they are in the two-party case.

Finally, we demonstrate that in the final step of the Whitewash protocol, the output release, that early termination by M^* or C^* is functionally the same as C^* returning an incorrect value for p_a^*. That is, APPLICATION will always detect an early termination, and will always detect an incorrect value of p_a^* except for a negligible probability.

Lemma 33. *The probability of catching a malformed p_a^* is computationally indistinguishable from catching early termination.*

Proof. Since the output $f_a(x^*, y) \oplus p_a^*$ was generated by a good circuit and C^*s input to that circuit was consistent, then the output of the hash $\mathbf{H} \cdot p_a^*$ is correctly computed. Thus, let $\overline{p}_a$ be the value that C^* returns during the output release. By the guarantees of a 2-universal hash, the probability that $\mathbf{H} \cdot p_a^* = \mathbf{H} \cdot \overline{p}_a$ is negligible. □

Given these changes to the simulator S_1 in shelat and Shen's proof, the modified simulator S_{MC} proves Theorem 2 (without the fair release guarantee) when the mobile device and CLOUD are malicious and colluding (scenario 3).

Table 3: Operations required on the mobile device by three outsourcing protocols. Here, SYM is the symmetric cryptographic operations, GROUP is the group algebraic operations, OT is the oblivious transfers, and CT is whether the protocol requires a coin toss. Recall that k is the security parameter, λ is the number of circuits generated, x is the mobile device's input, and y is the application server's input.

Protocol	SYM	GROUP	OT	CT
CMTB	$\|x\|$	$\frac{2\lambda}{5}(\|y\|+1)$	k	yes
Salus	$\frac{2\lambda}{5}(\|x\|+\|y\|+\|f(x,y)\|)$	-	-	yes
WW	$\lambda(\|x\|+\frac{2}{5}\|f_m(x,y)\|)$	-	-	no

4.5 Comparison with previous outsourcing protocols

In this section, we compare the asymptotic complexity and security guarantees of the Whitewash protocol to two previous outsourcing techniques: the protocol developed in Chapter 3, which we call "CMTB", and the Salus framework developed by Kamara et al. [92]. We refer to our Whitewash protocol as WW.

When examining the complexity of each protocol, recall that one of our main goals is to optimize the efficiency on the mobile device. Thus, we examine the number of operations each protocol requires on the mobile device itself. When compared to the underlying shelat-Shen protocol, Whitewash adds extra input values from the Cloud, but does not add any steps to the computation that increase the complexity of operations performed on the application server or the Cloud. Thus, for a discussion of the application server and Cloud protocol complexity, we refer the reader to the original work by shelat and Shen [151].

4.5.1 Comparison to CMTB

The underlying two-party computation protocols of Whitewash and CMTB follow similar structures in terms of the security checks that are performed. However, Kreuter, shelat, and Shen's (KSS) protocol [103], which underlies CMTB, uses a number of algebraic operations to perform input consistency checks and output proofs of consistency. The protocol developed by shelat and Shen [151], which underlies Whitewash, removes these expensive cryptographic primitives in favor of constructions that use only efficient, symmetric-key operations. In addition to the improvements to the underlying protocol, Whitewash outsources the generation side of two-party computation, while CMTB outsources the evaluation side.

In CMTB, since neither the mobile device or the Cloud could garble inputs before computation, a specially designed Outsourced Oblivious Transfer (OOT) protocol is necessary to deliver the mobile device's inputs to the evaluating Cloud in a secure, privacy-preserving manner. By swapping roles in the Whitewash protocol, we allow the mobile device to garble its own inputs, removing the need for an OT protocol to be performed from the mobile device. While Whitewash still requires OTs between the Cloud and the evaluating party, these operations can be parallelized, while the OOT protocol acts as a non-parallelizable bottleneck in computation.

4.5.1.1 Asymptotic Complexity

Table 3 shows this complexity for both Whitewash and CMTB. Note that for the mobile device, Whitewash requires significantly more symmetric key operations for garbling its own input and verifying the correctness of its output. By contrast, the OOT protocol in CMTB requires very few symmetric key operations, but requires several instantiations of an oblivious transfer. In addition, CMTB requires that the mobile device check the application server's input consistency and verify the correctness of the output using algebraic operations (e.g., modular exponentiations and homomorphic operations). Considering the fact that modular exponentiation is significantly more costly than symmetric key operations, removing these public key operations from the phone is a significant efficiency improvement for Whitewash. We also note that CMTB requires a two-party fair coin toss at the mobile device, which is not required by Whitewash.

4.5.1.2 Security Guarantees

The removal of the OOT protocol in Whitewash not only increases its efficiency when compared to CMTB, it also allows for stronger security guarantees. In CMTB, security was only possible if none of the parties collude, since the mobile device possessed information that would allow the Cloud to recover both input wire labels for all of the mobile input wires after the OOT. If the mobile device and Cloud collude in the Whitewash protocol, it simply removes the guarantee of fair release and makes the protocol equivalent to the underlying two-party computation protocol. Thus, the only guarantee lost is that of fair release at the

end of the protocol, since a colluding mobile device and Cloud may not release the one-time pad used to blind the evaluating party's output. We believe that this represents a more realistic security setting, since the mobile device is paying for the assistance of the Cloud and may collude.

4.5.2 Comparison to Salus

When considering the operations performed on the mobile device, the Salus protocol and the Whitewash protocol both make the mobile device responsible for generating circuit randomness and garbling its own inputs. However, the Whitewash protocol requires an added proof of output consistency that is not included in Salus. While this proof adds some complexity to the protocol, it allows Whitewash to handle functions where both parties get different output values, while Salus is designed to handle functions with a single, shared output value. In addition, the Whitewash protocol outsources the generation of the garbled circuit, while the malicious secure Salus protocol outsources the evaluation. By swapping the roles of the outsourced task and adding in consistency checks at the evaluating party, the Whitewash protocol guarantees security in a stronger adversarial model.

4.5.2.1 Asymptotic Complexity

Both the Whitewash and Salus protocols use only efficient, symmetric key operations, but there is a slight tradeoff in the number of operations required (Table 3). Salus only requires operations for the $\frac{2\lambda}{5}$ evaluated circuits, but requires those operations for each bit of both party's inputs and the shared output. By contrast, Whitewash requires that the mobile device's input be committed for all λ circuits generated, but then only requires correctness proof of the output wires on the $\frac{2\lambda}{5}$ evaluated circuits. When the application server's input is significantly longer than the mobile device's, this will cause the Salus protocol to be less efficient than Whitewash. However, in the average case where both inputs are approximately the same length, this will mean that Whitewash requires more operations. This small tradeoff in efficiency is justified by the fact that Whitewash provides security in a stronger adversarial model than Salus. We also note that Salus requires a two-party fair coin toss before the protocol begins, which is not required by Whitewash.

Table 4: Input size and circuit size for all test circuits evaluated.

Circuit	Input Size (Bits)	Total Gates		Non-XOR Gates	
		KSS	PCF	KSS	PCF
HAM (1600)	1,600	24,379	32,912	17,234	6,375
HAM (16384)	16,384	262,771	376,176	186,326	101,083
MAT (3x3)	288	424,748	92,961	263,511	27,369
MAT (5x5)	800	1,968,452	433,475	1,221,475	127,225
MAT (8x8)	2,048	8,067,458	1,782,656	5,006,656	522,304
MAT (16x16)	8,192	64,570,969	14,308,864	40,076,631	4,186,368
DIJK 10	112/1,040	259,232	530,354	118,357	291,490
DIJK 20	192/2,080	1,653,380	2,171,088	757,197	1,192,704
DIJK 50	432/5,200	22,109,330	13,741,514	10,170,407	7,549,370
RSA-256	256/512	934,092,960	673,105,990	602,006,981	235,925,023

4.5.2.2 Security Guarantees

The Salus protocol provides equivalent security guarantees to CMTB, guaranteeing security when none of the parties are colluding. This is a result of outsourcing the evaluation to the Cloud while allowing the mobile device to generate circuit randomness. If the mobile device colludes with the Cloud, they can trivially recover all of the other party's inputs. By outsourcing the generation of the garbled circuit and adding in additional consistency checks at the evaluating party, Whitewash guarantees security under this type of collusion. As stated above, the only guarantee lost is that of fair output release, which ultimately reduces Whitewash to the security of the underlying two-party computation protocol.

4.6 Performance Evaluation

Our protocol significantly expands upon the implementations of the PCF garbled circuit generation technique [104] and shelat and Shen's garbled circuit evaluation protocol [151]. For experimental comparison to previous protocols, we used the code implementation of the outsourcing protocol from Chapter 3, as well as an Android port of the two-party garbled circuit protocol developed by Kreuter, shelat, and Shen [103]. We would like to thank the authors of [103, 104, 151] for making their code available and for assisting us with our evaluation.

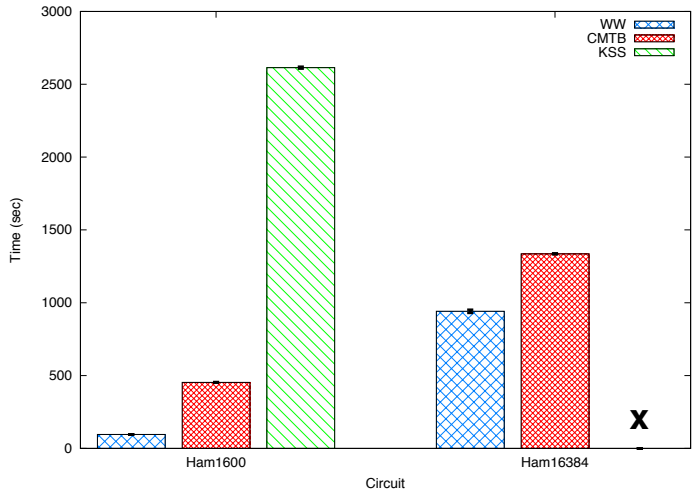


Figure 15: Execution time (ms) for Hamming Distance with input sizes of 1,600 and 16,384 bits for $\lambda = 256$ (note: log scale). Note that without outsourcing, only very small inputs can be computed over. Additionally, even for a large number of input bits, performing OTs on the servers still produces a faster execution time.

4.6.1 Test Environment

For evaluating our test circuits, we perform our experiments with a single server performing the role of Cloud and Application server, communicating with a mobile device over an 802.11g wireless connection. The server is equipped with 64 cores and 1TB of memory, and we partition the work between cores into parallel processing nodes using MPI. The mobile device used is a Samsung Galaxy Nexus with a 1.2 GHz dual-core ARM Cortex-A9 processor and 1 GB of RAM, running Android 4.0.

The large input sizes examined in the Hamming Distance trials required us to use a different testbed. For inputs as large as 16,384 bits, the phone provided by the above computing facility would overheat and fail to complete computation. Because the gate counts for Hamming Distance are significantly smaller than the other test circuits, we were able to run these experiments on a local testbed. We used two servers with Dual Intel Xeon E5620 processors, each with 4 hyper-threaded cores at 2.4 GHz each for the Cloud and the Application server. Each server is running the Linux kernel version 2.6, and is connected by a VLAN through a 1 Gbps switch. Our mobile device is a Samsung Galaxy Note II with a 1.6 GHz quad-core processor with ARM Cortex A9 cores and 2 GB of

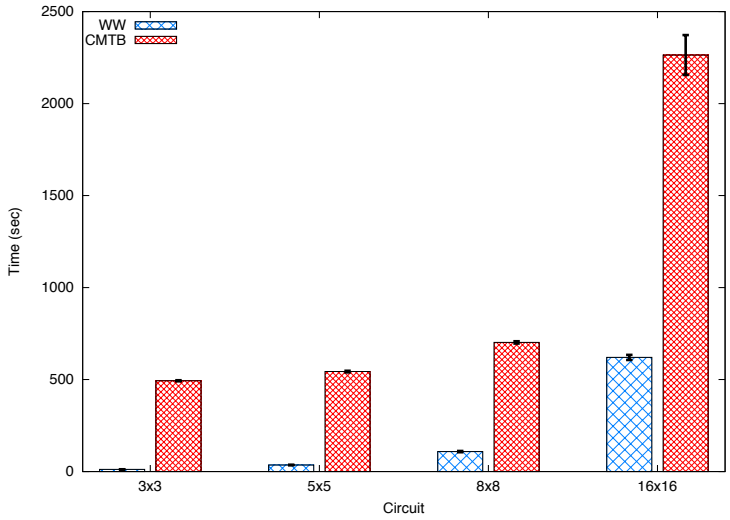


Figure 16: Execution time (ms) for the Matrix-Multiplication problem with input size varying between 3×3 matrices and 16×16 matrices for $\lambda = 256$ (note: log scale). This figure clearly shows that the oblivious transfers, consistency checks, and larger circuit representations of CMTB add up to a significant overhead as input size and gate count increase. By contrast, Whitewash requires less overhead and increases more slowly in execution time as gate counts and input size grow.

RAM, running the Android operating system at version 4.1. The phone connects to the two servers through a Linksys 802.11g wireless router with a maximum data rate of 54 Mbps. While this test environment represents optimistic connection speeds that may not always be available in practice, it allows us to consider the performance of the protocol without interference from variable network conditions, and mirrors the test environments used in previous work [103, 151]. For all experiments except RSA-256, we take the average execution time over ten test runs, with a confidence interval of 95%. For RSA-256, we ran 3 executions.

4.6.2 Experimental Circuits

To evaluate the performance of our protocol, we run tests over the following functions. We selected the following test circuits because they exercise a range of the two major variables that affect the speed of garbled circuit protocols: input size and gate counts. In addition, these programs are becoming somewhat standard test applications, having been used as benchmarks in a large amount of the related literature [103, 151, 104]. All of the programs are implemented with the algorithms used by Kreuter et al. [104] except for Dijkstra's

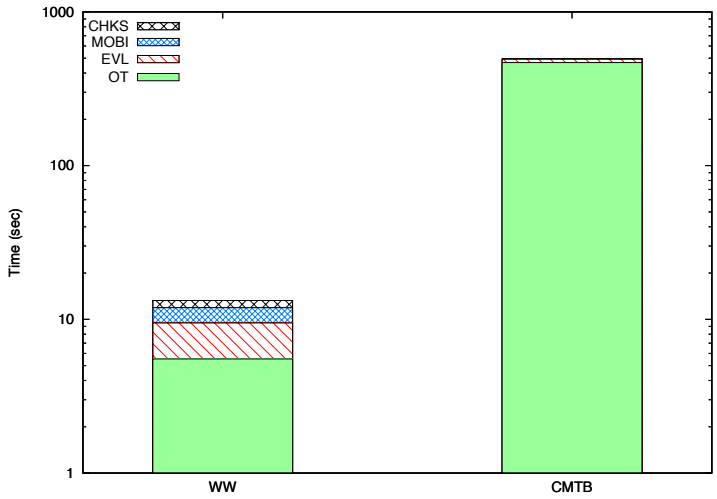


Figure 17: Microbenchmarking execution times (ms) for Whitewash and CMTB over the Matrix-Multiplication problem. We denote the total time spent in computation for Whitewash as "MOBI". Since the mobile device is linked with "CHKS" and "OT" in CMTB, we do not separate out the mobile time for that protocol. Notice the dominating amount of time required to perform oblivious transfers. Moving these operations off the mobile device removes a significant computation bottleneck.

algorithm, which matches the implementation used in Chapter 3:

- **Hamming Distance (HAM):** The Hamming Distance circuit accepts binary string inputs from both parties and outputs the number of locations at which those strings differ. This circuit demonstrates performance for a small number of gates over a wide range of input sizes. We consider input strings of length 1,600 bits and 16,384 bits.

- **Matrix Multiplication (MAT):** Matrix multiplication takes an $n \times n$ matrix of 32-bit integer entries from each party and outputs the result of multiplying the matrices together. This circuit demonstrates performance when both input size and gate count vary widely. We consider square matrix inputs where $n = 3, 5, 8$, and 16.

- **Dijkstra's Algorithm (DIJK):** This version of Dijkstra's algorithm takes an undirected weighted graph with a grid structure and a maximum node degree of four from the first party, and a start and end node from the second party. The circuit outputs the shortest path from the start node to the end node to the second party, and nothing to the first. For an n node graph, the graph description from the first party requires

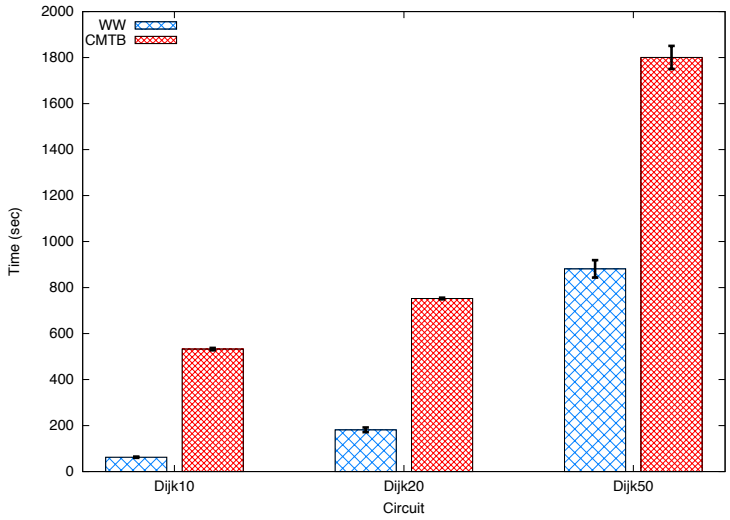


Figure 18: Execution time (s) for Dijkstra's algorithm with input sizes of $10, 20$, and 50 node graphs for $\lambda = 256$. This figure shows that the Whitewash protocol allows for computation that was only feasible to be executed in a close to practically useful time frame.

$104n$ input bits, while the start and end node descriptions require $8n + 32$ bits. We consider graphs with $n = 10, 20$, and 50 nodes. Due to an error in the PCF compiler, we were unable to compile a program for graphs larger than 50 nodes.

- **RSA Function (RSA):** The RSA function (i.e., modular exponentiation) accepts an RSA message from one party and an RSA public key from the other party and outputs the encryption of the input plaintext under the input public key. Specifically, one party inputs the modulus $n = pq$ for primes p and q, as well as the encryption key $e \in \mathbb{Z}_{\phi(n)}$. The other party inputs a message $x \in \mathbb{Z}_n^*$, and the circuit computes $x^e \pmod{n}$. This circuit demonstrates performance for small input sizes over very large gate counts. We consider the case where the input values x, n, and e are 256 bits each. While these are not secure parameters in practice, the function itself provides a complex circuit that is scalable on input size and useful for benchmarking our protocol.

For each test circuit, we consider the time required to execute and the bandwidth overhead to the mobile device. Table 6 shows the input size and gate counts for each test circuit, showing the exact range of values tested for these two circuit variables.

4.6.3 Execution Time

In all experiments, the efficiency gains of removing oblivious transfers and public key operations are immediately apparent. To examine how Whitewash compares to generating garbled circuits directly on the mobile device, we considered Hamming Distance as a simple problem (Figure 15). Even with a relatively small gate count, garbling the circuit directly on the mobile device is only possible for the small input size of 1,600 bits. Whitewash is capable of executing this protocol in 96 seconds, while running the computation directly on the mobile device takes 2,613 seconds, representing a 96% performance improvement through our outsourcing scheme. For the very large input size of 16,384 bits, computation directly on the mobile device ceases to be possible. When comparing to CMTB, this circuit further illustrates the cost of oblivious transfers on the mobile device. Even with the significantly reduced number of OTs allowed by the OOT protocol in CMTB (80 OTs), performing 16,384 malicious secure oblivious transfers between two servers in Whitewash still runs 30% faster than CMTB.

The matrix-multiplication circuit provides a good overview of average-case garbled circuit performance, as it represents a large range of both gate counts and size of inputs. For the input size of a 3×3 matrix, the Whitewash protocol runs in an average of 12 seconds, while CMTB requires 493 seconds, representing a 98% improvement (see Figure 16). Upon inspecting the micro benchmarking breakdown of each protocol's execution in Figure 17, we observe a significant speedup simply by moving oblivious transfers off of the mobile device. Even though the number of OTs required by CMTB is essentially constant based on their application of the Ishai OT extension, performing standard malicious secure oblivious transfers in parallel between the servers is much more efficient than requiring that the phone perform these costly operations. In addition, if we examine the amount of execution time where the phone participates in Whitewash, we see that the mobile device ("MOBI" in Figure 17), takes around 1 second, and is idle during the majority of computation. By contrast, both the OT and consistency check phases of CMTB require the mobile device to participate in a significant capacity, totaling almost 8 minutes of the computation. Having the phone perform as little work as possible means that the Whitewash protocol performance

is nearly equivalent to running the same computation between two server-class machines.

To examine the performance of Whitewash for a more practical application, we considered the Dijkstra's algorithm circuit used to implement privacy-preserving navigation in Chapter 3. They point out that this application, which has uses from military convoys to industrial shipping routes, is a significant first step in providing privacy for the growing genre of location-based mobile applications. Unfortunately, the PCF compiler does not optimize the Dijkstra's circuit as well as the previous experimental programs, which is evident in Table 6. In the 10 and 20 node graphs, the PCF compiler even produces a *larger* circuit than the compiler used by KSS. However, despite evaluating larger circuits, the Whitewash protocol still outperforms CMTB in execution time, running 88%,76%, and 51% faster in the $10, 20$, and 50 node cases respectively (shown in Figure 18). As circuit compilers continue to improve and produce smaller circuits, the performance gains of the Whitewash protocol will be even larger. In this experiment, we also noticed that because Whitewash evaluates and checks circuits simultaneously, it created contention for the network stack in our test server. In a truly distributed environment where each server node has dedicated network resources, the highly parallelizable structure of shelat and Shen's protocol would allow Whitewash to execute faster. Given that Whitewash can execute Dijkstra's algorithm obliviously on the order of minutes, it allows computation considered only feasible for previous schemes to be performed in a nearly practical execution time.

The previous experiments clearly show that outsourcing is necessary to run circuits of any practical size. For our final test circuit, we consider an extremely complex problem to demonstrate the ability of outsourcing protocols in the worst-case. The RSA-256 circuit evaluated by Kreuter et al. in [104] and shelat and Shen in [151] represents one of the largest garbled circuits ever evaluated by a malicious secure protocol. For the RSA-256 problem, Whitewash completed the computation in 515 minutes. CMTB was unable to complete one execution of the protocol. A large part of this efficiency improvement results from the underlying protocol of Whitewash, which uses only symmetric-key operations outside of the oblivious transfers between the servers. The reduced non-XOR gate counts and

Table 5: Bandwidth measures for all experiment circuits. Note that there is as much as a 84% reduction in bandwidth when using the Whitewash protocol.

	Bandwidth (MB)			Reduction Over	
Circuit	WW	CMTB	KSS	CMTB	KSS
HAM (1600)	23.56	41.05	240.33	42.62%	90.20%
HAM (16384)	241.02	374.03	x	35.56%	x
MAT (3x3)	4.26	11.50	x	62.97%	x
MAT (5x5)	11.79	23.04	x	48.82%	x
MAT (8x8)	30.15	51.14	x	41.05%	x
MAT (16x16)	120.52	189.52	x	36.41%	x
DIJK 10	1.67	20.21	x	91.73%	x
DIJK 20	2.85	35.28	x	91.93%	x
DIJK 50	6.38	80.49	x	92.08%	x
RSA-256	3.97	x	x	x	x

more compact circuit representation of the PCF compiler also contribute to this improvement. Ultimately, because Whitewash ensures that the phone participates minimally in the protocol, it no longer acts as a bottleneck on computation. We essentially reduce performance of our outsourcing protocol to that of the underlying two-party protocol, allowing this technique for outsourcing to benefit as more improvements are made in non-outsourced garbled circuit protocols. In addition, this minimal level of interactivity allows us to run these protocols with 256 circuits, equivalent to a security parameter of approximately 80-bit security, which is agreed by the research community to be an adequate security parameter. Finally, the phone is only active for a few seconds during this large computation, keeping its system resources free for other user applications (or to preserve battery power) while the servers complete the computation. *This shows that Whitewash is capable of evaluating the same circuits as the most efficient desktop-based garbled circuit protocols with a minimal overhead cost.*

4.6.4 Network Bandwidth

The Whitewash protocol not only improves the speed of execution when outsourcing garbled circuit computation, it also significantly reduces the amount of bandwidth required by the mobile device to participate in the computation. Table 9 shows the bandwidth used by the mobile device for each test circuit. In the best case, for Dijkstra's algorithm over 50 node graphs, we observed a 92% reduction in bandwidth between Whitewash and CMTB.

This is a result of the mobile device not performing OTs and only sending relatively small symmetric-key values instead of algebraic elements for consistency checks. For all test circuits, we observed a small decrease in the amount of improvement between the two protocols as the input size increased. This is because the number of commitments sent by the phone in Whitewash increases as the size of the input grows, while CMTB performs a fixed number of OTs as the input size increases. However, the oblivious transfers still require a significant enough amount of bandwidth to make removing them the most efficient option. When comparing to not outsourcing garbled circuit generation, the cost of oblivious transfers and sending several copies of the garbled circuit to the evaluator quickly adds up to a significant bandwidth cost. For the smallest circuit evaluated, outsourcing the circuit garbling reduces the required amount of bandwidth by 90%. The importance of these bandwidth reductions is further highlighted when considering mobile power savings. With data transmission costing roughly 100 times as much power as computation on the same amount of data, any reduction in the bandwidth required by a protocol implies a critical improvement in practicality.

One challenge encountered during the implementation of the Whitewash protocol was the extensive use of hardware-specific functions used to implement commitment schemes in shelat and Shen's code. Rather than try to port this code over to Android, which would require significant development of hardware-specific libraries, we chose to implement the protocol in an equivalent manner by having the Cloud generate the part of the commitments which requires these functions and send them to the mobile device. The mobile device then finishes generating the commitments that match its input and forwards them to the evaluator. Our proofs of security remain valid even with this small protocol modification. Our preliminary implementation using instructions specific to the ARM architecture has shown that we could further reduce the measured bandwidth values by over 60%. With already significant bandwidth reductions from previous outsourcing schemes, our protocol will see further improvements as mobile hardware begins to incorporate more machine-specific libraries.

4.7 Conclusion

Outsourcing garbled circuit protocols has been shown to significantly improve the efficiency of SMC protocols used in smartphone applications while maintain equivalent privacy guarantees to non-outsourced protocols. However, the cryptographic primitives used in previous protocols cause the mobile device to act as a bottleneck in computation. Our Whitewash protocol presents a new scheme that eliminates the most costly operations, including oblivious transfers, from the mobile device. By requiring that the mobile device instead produce the randomness required for circuit generation, we significantly reduce the number of algebraic group operations and communication rounds for the mobile device. At the same time, we bolster the security guarantees against certain types of collusion, yielding a more secure protocol than any other in this space. Our performance evaluation shows average gains of 75% for execution time and 60% for bandwidth over the previous outsourcing protocol. These improvements allow large circuits representing practical applications to be computed efficiently from a mobile device. As a result, we show that outsourcing has the potential to make garbled circuit protocols as efficient for mobile devices as they are for server-class machines.

CHAPTER V

BLACK BOX SMC OUTSOURCING

5.1 Introduction

Our Whitewash protocol demonstrated that garbled circuit SMC can be outsourced with a very small amount of computation on the mobile device and minimal overhead additions to the server-side computation. Furthermore, that protocol moved the bottleneck for computation off of the mobile device and onto the underlying SMC primitive. Because of this, to significantly improve the efficiency of mobile SMC protocols, it is now necessary to improve the efficiency of the underlying SMC technique. Unfortunately, both of our previous protocols are build on fixed underlying SMC techniques. As new techniques and models for garbled circuit computation are developed, these previous protocols cannot be directly applied to provide an outsourced version, which limits the forward applicability of these protocols.

In this chapter, we develop a technique for outsourcing secure two-party computation for *any* two-party SMC technique. Rather than adding consistency checks to the outsourcing protocol, we add a small amount of overhead to the evaluated function itself, then evaluate this augmented functionality with any malicious secure two-party SMC protocol. This tradeoff allows for an outsourcing scheme that relies on the underlying two-party protocol in a black box manner, meaning the protocol can be swapped for any other protocol meeting the same definition of security This makes the task of securely incorporating newly developed SMC techniques trivial. This protocol enables mobile devices to participate in *any secure two-party SMC protocol* with minimal cost to the device and with nominal overhead to the servers running the computation. Specifically, we make the following contributions:

- **Develop a black box outsourcing protocol:** We develop a novel outsourcing technique for lifting any two-party SMC protocol into the two-party outsourced setting.

To do this, we add a small amount of overhead to the function being evaluated to ensure that none of the inputs are modified by malicious participants. This technique of augmenting the evaluated circuit has been successfully used in other SMC protocols to balance performance with security guarantees [83, 109, 151]. In addition, we leverage the non-collusion assumption used throughout the related work to produce an output consistency check that incurs trivial overhead. While this approach slightly increases the cost of evaluation, it minimizes the computation and bandwidth required by the mobile device.

- **Prove security for any underlying two-party SMC protocol:** We provide simulation proofs of security to demonstrate that our protocol is secure in the malicious threat model. The only requirement of the underlying two-party SMC protocol is that it satisfy the canonical ideal/real world simulation definition of security against malicious adversaries [66]. This allows *any* future SMC protocols that are developed to be used in a plug-&-play manner with our outsourcing technique.

- **Implement and evaluate the overhead cost of the outsourcing operations:** Using the garbled circuit two-party SMC protocol of shelat and Shen [151], we implement our protocol and evaluate the complete overhead cost of outsourcing. Rather than compare to our previous outsourcing schemes, we instead measure the overhead incurred by augmenting the desired functionality, as well as the input and output preparation and checking. This measurement of cost better represents the value of the scheme, as a direct comparison to previous outsourcing protocols would drastically change depending on the underlying two-party SMC protocol implemented in our black box scheme. Our results show that for large circuits, black box outsourcing incurs negligible overhead (i.e., the confidence intervals for outsourced and server only execution intersect) in evaluation time and in bandwidth required when compared to evaluating the unmodified function. To demonstrate the practical performance of our protocol, we develop a mobile-specific facial recognition application and analyze its performance.

This work was originally published at the International Conference on Cryptology and Network Security [35].

5.2 Underlying techniques

Outsourced two-party SMC protocols are designed to allow two parties of asymmetric computational capability to engage in a privacy-preserving computation with the assistance of an outsourcing party. We consider the situation where a mobile device possessing limited computational resources wishes to run an SMC protocol with an application server or other well-provisioned entity. To allow this, outsourcing protocols move the majority of the costly operations off of the mobile device and onto a Cloud provider *without* revealing to the Cloud either party's input or output to the computation. These protocols aim to provide security guarantees of privacy and correctness, and also attempt to minimize the computation required at the mobile device while still maintaining efficiency between the application server and the Cloud. To meet these goals in the outsourced setting, a number of careful security assumptions must be made.

5.2.1 Two-party SMC security

Our black box protocol is based on the execution of a two-party SMC protocol to obliviously compute the result. We make no assumptions about the techniques used or structure of this underlying protocol except that it meets the canonical definition of security against malicious adversaries using the ideal/real world paradigm [66]. Informally, this states that for any adversary participating in the two-party SMC protocol, there exists a simulator in an ideal world with a trusted third party running the computation where the output in both worlds is computationally indistinguishable. In this definition, the simulator in the ideal world is given oracle access to the adversary in the real world. Particularly in the two-party setting, there are a few caveats that must be assumed to make this definition feasible, and must be considered when designing an outsourced protocol that uses a two-party protocol in a black box manner.

First, it is known that two-party protocols cannot fully prevent early termination. In any execution, one party will receive their output of computation before the other party

does. While certain techniques have been developed to partially solve this problem, there is no complete solution. While other outsourcing protocols have added in a fair-release guarantee, this guarantee comes at a cost. Either the protocol must provide additional commitments not guaranteed in a standard two-party protocol [92], or the protocol must incorporate additional costly MAC operations to ensure the output is not tampered with [92, 127]. However, our black box protocol shows that if we treat the outsourced model like a standard two-party execution where fair release is not guaranteed, we can reduce the output consistency check to a single comparison on the mobile device. This allows the application server to recover its input first and potentially disrupt the mobile device's output, but mirrors the two-party execution guarantees exactly. Thus, our protocol optimizes execution overhead by not assuming a fair output release.

Second, it is possible that a malicious party can provide arbitrary input to the computation that may or may not correspond to their "real" input. While we cannot control what another party provides as input to the computation, this potential behavior must be handled by the definition of security. To handle this, the simulator in the ideal world, which has oracle access to the adversary in the real world, must not only be able to simulate the adversary's view of the protocol. Upon running the adversary with a given input, the simulator must also be able to recover the actual input used by the adversary. In our proofs of execution, the ideal world will invoke this simulator often as a mechanism to recover the adversary's input before initiating computation with the trusted third party. This ensures that the output in both worlds is indistinguishable.

Given these assumptions, a secure two-party SMC protocol provides two guarantees. The first is privacy, which means that a malicious adversary cannot learn anything about the other party's input or output value beyond what is revealed by his own output value. The second guarantee is correctness. This implies that even in the presence of a malicious adversary, the output of the protocol will be the correct output of the agreed upon function except with negligible probability.

For a formal definition of security and further discussion, refer to [66].

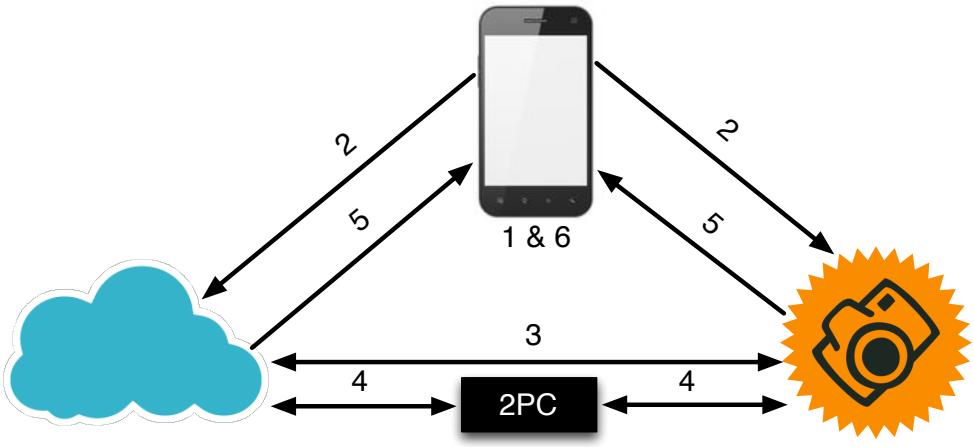


Figure 19: The complete black box outsourcing protocol. Note that the mobile device performs very little work compared to the application server and the Cloud, which execute a two-party SMC (2PC) protocol.

5.2.2 Security definition and non-collusion

We follow the security definition first established by Kamara et al. [92] but specified for the two-party scenario as in Chapter 3. We slightly alter the definition and allow for the possibility of early termination by one of the parties in the ideal world following the standard two-party definition [66]. This captures the fact that our black box construction does not provide fair release as in our first two outsourcing protocols. Furthermore, we make the same non-collusion assumption as Whitewash in Chapter 4, that the Cloud and the application server do not collude against the mobile device but the mobile device and the Cloud may collude against the application server.

5.3 Protocol

In this section, we formally define our black box outsourcing protocol. For a graphical representation, see Figure 19.

5.3.1 Participants

Our protocol again uses the same participants as in Section 3.3.1. However, in this variation, Application and Cloud will be responsible for executing a standard two-party SMC

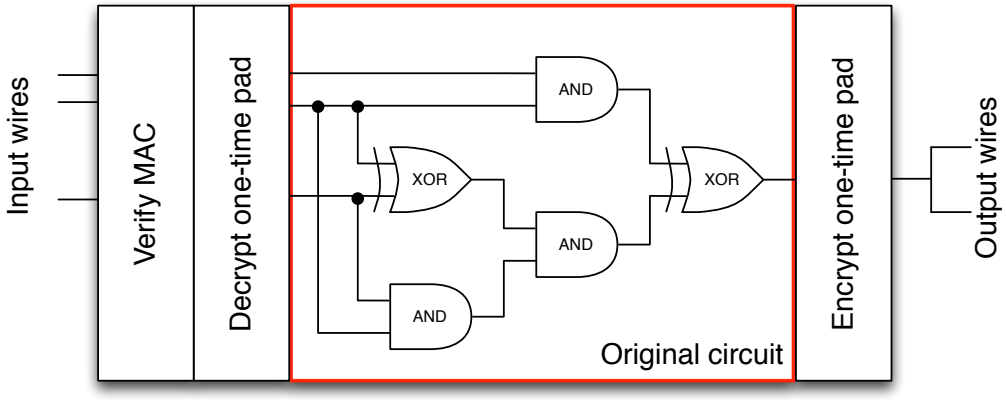


Figure 20: The process of augmenting a circuit for outsourcing. The original circuit is boxed in red. Essentially, we require that the mobile device's input be verified using a MAC and decrypted using a one-time pad before it is input into the function. After the result is computed, it must be re-encrypted using a one-time pad and delivered to *both* parties to guarantee that the mobile device will detect if either party tampers with the result.

protocol. MOBILE prepares some input that is used by both of the parties executing the SMC protocol and is verified during the black box computation.

5.3.2 Overview

The outsourcing protocol can be informally broken down as follows: first, the mobile device prepares its input by encrypting it and producing a MAC tag for verifying the input is not tampered with before it is entered into the computation. Since the application server and Cloud are assumed not to collude, one party receives the encrypted input, and the other party receives the decryption key. Both of these values are input into the secure two-party computation, and are verified within the secure two-party protocol using the associated MAC tags (see Figure 20). If the check fails, the protocol outputs a failure message. Otherwise, the second phase of the protocol, the actual evaluation of the SMC program, takes place. The third and final phase encrypts and outputs the mobile device's result to both parties, who in turn deliver these results back to the mobile device. Intuitively, since our security model assumes that the application server and the Cloud are never simultaneously malicious, at least one of these two will return the correct result to the mobile device. From this, the mobile will detect any tampering from the malicious party by a discrepancy in these

returned values, eliminating the need for an output MAC. If no tampering is detected, the mobile device then decrypts the output of computation.

5.3.3 Protocol

Common Input: All parties agree on a computational security parameter k, a message authentication code (MAC) scheme $(Gen(), Mac(), Ver())$, and a malicious secure two-party computation protocol $2PC()$. All parties agree on a two-output function $f(x, y) \rightarrow f_m, f_a$ that is to be evaluated.

Private Input: MOBILE inputs x while APPLICATION inputs y. We denote the bit length of a value as $|x|$ and concatenation as $x||y$.

Output: APPLICATION receives f_a and MOBILE receives f_m.

1. **Input preparation:** MOBILE generates a one-time pad k_{fm} where $|k_{fm}| = |f_m|$. Mobile then generates two MAC keys $v_a = Gen(k)$ and $v_c = Gen(k)$. Finally, MOBILE generates a one-time pad k_m where $|k_m| = |x| + |k_{fm}|$.

2. **Input delivery:** MOBILE encrypts its input as $a = (x||k_{fm}) \oplus k_m$. It then generates two tags $t_a = Mac(a||v_c, v_a)$ and $t_c = Mac(k_m||v_a, v_c)$. MOBILE delivers a, v_c, and t_a to APPLICATION and k_m, v_a, and t_c to CLOUD.

3. **Augmenting the target function (Algorithm 1):** All parties agree on the following augmented function $g(y, a, v_c, t_a; k_m, v_a, t_c)$ to be run as a two-party SMC computation:

   (a) If $Ver(a||v_c, t_a, v_a) \neq 1$ or $Ver(k_m||v_a, t_c, v_c) \neq 1$ output $\perp$.

   (b) Set $x||k_{fm} = a \oplus k_m$

   (c) Run the desired function $f_a, f_m = f(x, y)$

   (d) Set output values $o_a = f_a$ and $o_m = f_m \oplus k_{fm}$

   (e) Output $o_a||o_m$ to APPLICATION and o_m to CLOUD

4. **Two-party computation:** APPLICATION and CLOUD execute a secure two-party

Input : CLOUD inputs k_m, v_a, t_c and APPLICATION inputs y, a, v_c, t_a
Output: CLOUD receives o_m and APPLICATION receives $o_a||o_m$

if $Ver(a||v_c, t_a, v_a) \neq 1$ **then**
  return $\perp$
else if $Ver(k_m||v_a, t_c, v_c) \neq 1$ **then**
  return $\perp$
else
  $x, k_{fm} = a \oplus k_m$
  $f_m, f_a = f(x, y)$
  $o_a = f_a(x, y)$
  $o_m = f_m(x, y) \oplus k_{fm}$
end

Algorithm 1: The augmented function

computation protocol $2PC(g(); y, a, v_c, t_a; k_m, v_a, t_c)$ evaluating the augmented function.

5. **Output verification:** CLOUD delivers its output from the two-party computation, o_m to MOBILE. APPLICATION also delivers the second half of its output o'_m to MOBILE. MOBILE verifies that $o_m = o'_m$.

6. **Output recovery:** APPLICATION receives output $f_a = o_a$ and MOBILE receives output $f_m = o_m \oplus k_{fm}$

5.4 Security

Our black box outsourcing protocol is secure under the following theorem satisfying the security definition from Section 5.2:

Theorem 3. *The black box outsourced two-party protocol securely computes a function $f(x, y)$ in the following two corruption scenarios: (1) Any one party is malicious and non-cooperative with respect to the rest of the parties; (2) The Cloud and the mobile device are malicious and colluding, while the application server is semi-honest.*

Note that these scenarios correspond exactly with the corruption scenarios in Chapter 4, and that the previous protocols described in Kamara et al. [92] and Chapter 3 are only secure in corruption scenario (1).

5.4.1 Malicious Cloud or Application Server

The main idea behind the security in these two settings is that for whichever party is corrupted, we can rely on the other party to behave semi-honestly. Based on the security of the underlying two-party protocol, this ensures both that the augmented functionality is correctly evaluated and that the mobile device will receive unmodified output from one of the parties. Thus, the MAC on the input and the comparison of the output values prevents either party from modifying the Mobile device's private values. Furthermore, unlike the dual execution model by Huang et al. [83] where the output comparison leaks one bit of input, our output comparison is composed of two copies of the mobile output produced from a single, malicious-secure execution of the augmented circuit. Because of this, any discrepancy in the comparison only reveals that either the Cloud or Application Server tampered with the output prior to delivering it to the mobile device.

In the ideal world, the simulator works roughly as follows: begin the black box protocol with random inputs. Then, invoke the simulator for the underlying two-party scheme S_{2PC} to recover the input of the malicious party and delivers that input to the trusted third party. Finally, S_{2PC} simulates the output $f(x, y)$. After running all consistency verifications, the simulator either sends an early termination signal to the trusted third party or completes the protocol normally.

5.4.2 Malicious Mobile Device

Because the mobile device simply provides MAC tagged input and receives its output after executing the two-party protocol, there is very little it can do to corrupt the computation besides providing invalid inputs that would simply cause the computation to terminate early. The simulator in this scenario accepts the mobile device's prepared inputs. Given both the Cloud and the Application Server's halves of the mobile device's input, the simulator can recover the necessary input by decrypting the one-time pad. If either of the MAC tags does not verify or if the mobile device terminates early, the simulator also terminates. Otherwise, it invokes the trusted third party to receive $f(x, y)$ and returns the result to the mobile device.

5.4.3 Malicious Mobile Device and Cloud

In this scenario, the security of our black box protocol simply reduces to the security of the underlying two-party scheme. The simulator in the ideal world accepts the input from the Mobile Device, then invokes the simulator of the underlying two-party SMC scheme S_{2PC} to recover the values input by the Cloud. Using these values combined with the values provided by the Mobile Device, the simulator can recover the Mobile input. If any of the verification checks within the augmented functionality fail, the simulator terminates. Otherwise, it delivers the recovered input to the trusted third party, and finishes S_{2PC} delivering the output of computation correctly formatted using the one-time pads recovered from the Cloud's input by S_{2PC}.

5.5 Proof of Security

Here we provide the formal simulation proof of security for Theorem 3.

5.5.1 Malicious Mobile M^*

In the scenario where M^* can adopt an arbitrary malicious strategy, we construct a simulator S_M that, operating in the ideal world, can simulate M^*s view of a real world protocol execution and can recover M^*s input for delivery to the trusted third party. We construct this simulator and prove it secure with the following hybrid of experiments.

$Hyb1^{(M)}(k, x; r)$: This experiment is identical to $REAL^{(M)}(k, x; r)$ except that the experiment uses the combination of M^*s encrypted input a and k_m to recover the real input $x*$. It verifies the MAC tags t_a and t_c and aborts if either check fails.

Lemma 34. $REAL^{(M)}(k, x; r) \stackrel{c}{\approx} Hyb1^{(M)}(k, x; r)$

Proof. Since the experiment is controlling both Cloud and Application, it can simply decrypt the input x^* using the key k_m. In addition, since the experiment holds both the verification keys, the protocol will terminate in both experiments if the MAC tags are incorrectly constructed. □

$Hyb2^{(M)}(k, x; r)$: This experiment is identical to $Hyb1^{(M)}(k, x; r)$ except that the experiment passes x^* to the trusted third party, and returns the result $f(x^*, y) \oplus k^*_{fm}$ to M^*, where k^*_{fm} is recovered in the previous hybrid.

Lemma 35. $Hyb1^{(M)}(k, x; r) \overset{c}{\approx} Hyb2^{(M)}(k, x; r)$

Proof. Because both experiments use the input x^* for computing the result, the output of the function in both worlds is indistinguishable. Furthermore, the recovered output key allows the experiment to present the result to M^* exactly as it would be in a real world execution. □

Lemma 36. $Hyb2^{(M)}(k, x; r)$ *runs in polynomial time.*

Proof. This lemma follows trivially since a real world execution of the protocol runs in polynomial time and each intermediate hybrid adds only constant time operations. □

We conclude the proof by letting S_M execute $Hyb2^{(M)}(k, x; r)$. S_M runs M^* and controls CLOUD and APPLICATION. S_M terminates the ideal world execution if any consistency checks fail or if M^* terminates at any point, and outputs whatever M^* outputs at the end of the simulation. From Lemma 34-36, S_M proves Theorem 3 when MOBILE is malicious.

5.5.2 Malicious Application A^*

In the scenario where A^* can adopt an arbitrary malicious strategy, we construct a simulator S_A that, operating in the ideal world, can simulate A^*s view of a real world protocol execution and can recover A^*s input for delivery to the trusted third party. We construct this simulator and prove it secure with the following hybrid of experiments.

$Hyb1^{(A)}(k, x; r)$: This experiment is identical to $REAL^{(A)}(k, x; r)$ except that the experiment prepares the MOBILE input according to the two-party protocol simulator S_{2PC} instead of using the real MOBILE input. It then prepares the new input according to the protocol and delivers the encrypted input and MAC tags to A^*.

Lemma 37. $REAL^{(A)}(k, x; r) \overset{c}{\approx} Hyb1^{(A)}(k, x; r)$

Proof. Since the input is blinded by a one-time pad in both experiments, they are statistically indistinguishable. □

$Hyb2^{(A)}(k, x; r)$: This experiment is identical to $Hyb1^{(A)}(k, x; r)$ except that the experiment invokes the simulator of the two-party SMC protocol S_{2PC} instead of running the actual protocol. S_{2PC} is used to recover A^*s actual input y^*. After recovering the full input, If A^* tampers with MOBILE'S input, S_{2PC} simulates $\perp$ and the experiment terminates. Otherwise, the experiment delivers y^* to the trusted third party and simulates the output $f(x, y^*)$ concatenated with a random string o_{rm}.

Lemma 38. $Hyb1^{(A)}(k, x; r) \stackrel{c}{\approx} Hyb2^{(A)}(k, x; r)$

Proof. Based on the security definition of the underlying two-party SMC protocol, we know that a simulator exists that can simulate the protocol in a computationally indistinguishable way, as well as recover the input used by A^*. Based on the correctness guarantee of the two-party SMC protocol in conjunction with the unforgettability guarantee of the MAC protocol, it is computationally infeasible for A^* to modify MOBILE'S portion of the input. Finally, in both experiments the MOBILE output of the computation is blinded by a one-time pad, making the random output statistically indistinguishable from the real output. □

$Hyb3^{(A)}(k, x; r)$: This experiment is identical to $Hyb2^{(A)}(k, x; r)$ except that the experiment prevents the trusted third party from delivering input to the other party if A^* modifies the MOBILE output o_{rm} before returning it.

Lemma 39. $Hyb2^{(A)}(k, x; r) \stackrel{c}{\approx} Hyb3^{(A)}(k, x; r)$

Proof. Based on the correctness guarantee of the two-party SMC scheme and the fact that CLOUD is semi-honest in this scenario, then A^* will be caught in either experiment, and early termination will be the result. □

Lemma 40. $Hyb3^{(A)}(k, x; r)$ *runs in polynomial time.*

Proof. This lemma follows trivially since a real world execution of the protocol runs in polynomial time, the simulator S_{2PC} runs in polynomial time, and all other intermediate hybrid adds only constant time operations. □

We conclude the proof by letting S_A execute $Hyb3^{(A)}(k, x; r)$. S_A runs A^* and controls CLOUD and MOBILE. S_A terminates the ideal world execution if any consistency checks fail or if A^* terminates at any point, and outputs whatever A^* outputs at the end of the simulation. From Lemma 37-40, S_A proves Theorem 3 when APPLICATION is malicious.

5.5.3 Malicious Cloud C^*

In the scenario where C^* can adopt an arbitrary malicious strategy, we construct a simulator S_C that, operating in the ideal world, can simulate C^*s view of a real world protocol execution and can recover C^*s auxiliary input for delivery to the trusted third party. We construct this simulator and prove it secure with the following hybrid of experiments.

$Hyb1^{(C)}(k, x; r)$: This experiment is identical to $REAL^{(C)}(k, x; r)$ except that the experiment invokes the two-party SMC simulator S_{2PC}, providing random inputs for APPLICATION and recovering C^*s real input. Finally, simulate a random result o_r at the end of the two-party computation.

Lemma 41. $REAL^{(C)}(k, x; r) \stackrel{c}{\approx} Hyb1^{(C)}(k, x; r)$

Proof. Based on the security definition of the underlying two-party SMC protocol, we know that the simulator S_{2PC} can indistinguishably simulate the two-party execution and recover MOBILE'S MAC tagged one-time pad as input by C^*. Because in both experiments the output of the circuit is blinded by a one-time pad, the outputs in both cases are statistically indistinguishable. □

$Hyb2^{(C)}(k, x; r)$: This experiment is identical to $Hyb1^{(C)}(k, x; r)$ except that if the experiment finds from the recovered input that C^* modified the random key k_m, the experiment terminates.

Lemma 42. $Hyb1^{(C)}(k, x; r) \stackrel{c}{\approx} Hyb2^{(C)}(k, x; r)$

Proof. Based on the correctness guarantee of the two-party SMC scheme and the unforgettability of the MAC scheme, any change to k_m will cause the circuit to output $\perp$, and will

cause MOBILE to terminate except for a negligible probability. Thus, termination in both experiments in computationally indistinguishable. □

$Hyb3^{(C)}(k, x; r)$: This experiment is identical to $Hyb2^{(C)}(k, x; r)$ except that if the experiment aborts if C^* modifies the output string o_r.

Lemma 43. $Hyb2^{(C)}(k, x; r) \stackrel{c}{\approx} Hyb3^{(C)}(k, x; r)$

Proof. Because APPLICATION is semi-honest and will not tamper with MOBILE'S output, in both hybrids C^* will be caught for tampering with the output and result in an abort of the protocol. □

Lemma 44. $Hyb3^{(C)}(k, x; r)$ *runs in polynomial time.*

Proof. This lemma follows trivially since a real world execution of the protocol runs in polynomial time, the simulator S_{2PC} runs in polynomial time, and all other intermediate hybrid adds only constant time operations. □

We conclude the proof by letting S_C execute $Hyb3^{(C)}(k, x; r)$. S_C runs C^* and controls APPLICATION and MOBILE. S_C terminates the ideal world execution if any consistency checks fail or if C^* terminates at any point, and outputs whatever C^* outputs at the end of the simulation. From Lemma 41-44, S_C proves Theorem 3 when CLOUD is malicious.

5.5.4 Malicious Mobile and Cloud MC^*

In the final scenario, the colluding parties MC^* can adopt an arbitrary malicious strategy against APPLICATION. The simulator S_{MC} that proves security in this scenario is essentially the two-party SMC simulator S_{2PC} with one small change. Rather than completely recovering MC^*s input from the simulator, the experiment must combine the malicious MOBILE input $a^*||v_s^*||t_a^*$ with the input recovered by S_{2PC} to learn the real input x^* that is to be delivered to the trusted third party. Once this real input is retrieved, it simulates the result $f(x^*, y)$ exactly as S_{2PC} does. Since the added operations are constant time and S_{2PC} runs in polynomial time, we have that S_{MC} proves Theorem 3 when both MOBILE and CLOUD are malicious and colluding. Note that, as in the underlying two-party SMC scheme,

Table 6: A comparison of the original function size to the augmented outsourcing circuit.

Program Name	SS13 Total	BB Total	Increase	SS13 Non-XOR	BB Non-XOR	Increase
Dijkstra10	259,232	456,326	1.8x	118,357	179,641	1.5x
Dijkstra20	1,653,542	1,949,820	1.2x	757,197	849,445	1.1x
Dijkstra50	22,109,732	22,605,018	1.0x	10,170,407	10,324,317	1.0x
MatrixMult3x3	424,748	1,020,196	2.4x	161,237	345,417	2.1x
MatrixMult5x5	1,968,452	3,360,956	1.7x	746,977	1,176,981	1.6x
MatrixMult8x8	8,069,506	11,354,394	1.4x	3,060,802	4,075,082	1.3x
MatrixMult16x16	64,570,969	77,423,481	1.2x	24,494,338	28,458,635	1.2x
RSA128	116,083,727	116,463,648	1.0x	41,082,205	41,208,553	1.0x

this scenario does *not* guarantee that the output will be released fairly to Application. However, it does guarantee privacy and correctness of the output.

5.6 Performance Evaluation

To demonstrate the practical efficiency of our black box outsourcing protocol, we implemented the protocol and examined the actual overhead incurred by the overhead operations. We initially considered comparing our black box protocol to existing implementations of outsourcing protocols such as Salus [92] or the protocols described in the previous chapters. However, these existing protocols are built on fixed underlying SMC techniques. As new protocols for two-party SMC are developed, the plug-and-play nature of our protocol allows for these new techniques to be applied, which would provide a different comparison for each underlying protocol. Instead, we chose to compare the overhead execution costs of our black box protocol to performing the same computation in the underlying two-party protocol. Because the mobile device computation requires seconds or less to execute, we focus our attention on the cost at the two executing servers. This performance analysis demonstrates two key benefits of our protocol. First, it gives a rough overhead cost for an entire class of two-party SMC protocols (in our case, garbled circuit protocols). Second, it allows us to demonstrate that our outsourcing technique allows a mobile device with restricted computational capability to participate in a privacy-preserving computation in approximately the same amount of time as the same computation performed between two servers. Essentially,

we show that our protocol provides a mobile version of any two-party SMC protocol with nominal overhead cost to the servers. This is a novel evaluation methodology not used to evaluate previous black box SMC constructions, and provides a more intuitive estimate for performance when applying a new underlying SMC construction.

5.6.1 System Design

Our implementation of the black box outsourcing protocol uses the two-party garbled circuit protocol developed by shelat and Shen [151] as the underlying two-party SMC protocol. We selected this protocol because it is among the most recently developed garbled circuit protocols and it has the most stable public release. We emphasize that it is possible to implement our outsourcing on *any* two-party SMC protocol, such as the recent protocols developed to reduce the cost of cut-and-choose [79, 109]. We implement our MAC within the augmented circuit using AES in cipher-block chaining mode (CBC-MAC), as the AES circuit is well-studied in the context of garbled circuit execution. This MAC implementation adds an invocation of AES per 128-bit block of input. Using the compiler developed by Kreuter et al. [103], the overhead non-XOR gate count in the augmented circuit based on input size is $(\frac{|x|15686}{128})$ for input x. We provide exact gate counts with overhead measurements for each tested application in Table 6.

5.6.1.1 Testbed

Our experiments were run on a single server equipped with 64 cores and 1 TB of RAM. For each execution, the application server and cloud were run as 32 processes communicating using the Message Passing Interface (MPI) framework. The mobile device used was a Samsung Galaxy Nexus with a 1.2 GHz dual-core ARM Cortex-A9 processor and 1 GB of RAM, running Android version 4.0. The mobile device communicated with the test server over an 802.11n wireless connection in an isolated network environment. We ran each experiment 10 times and averaged the results, providing 95% confidence intervals in all figures.

Table 7: The total operations and bandwidth required at the mobile device. Recall that $|x|$ is the length of the mobile input in bits, k is the security parameter, and $|o_m|$ is the length of the mobile output in bits. When measured with the total protocol execution time, these operations are lost in the confidence intervals.

Symmetric	Asymmetric	Bandwidth (bits)
9	0	$2(\|x\| + 2k) + 4(\|o_m\|)$

5.6.1.2 Test applications

We selected a representative set of test applications from previous literature [103, 151, 104] to examine the performance of our protocol over varying circuit and input sizes. We use all applications as implemented by Kreuter et al. [103] except for Dijkstra's algorithm which is implemented as in Chapter 3.

1. Dijkstra: this application accepts a weighted graph from one party and two node indices from the other party (i.e., start and end nodes), and calculates the shortest path through the graph from the start to the end node. We consider n-node graphs with 16 bit edge weights, 8 bit node identifiers, and a maximum degree of 4 for each node. We chose this problem as a representative application for the mobile platform.

2. Matrix Multiplication: this application accepts a matrix from both parties and outputs the matrix product. We consider this application for input size n, where each matrix is an $n \times n$ matrix of 32-bit integers. This test application demonstrates protocol behavior for increasing input sizes.

3. RSA: this application accepts a modulus N and an exponent e from one party, and a message x from the other party, and computes the modular exponentiation $x^e \mod N$. We consider input values where each value is 128 bits in length. While this is certainly too short for secure practical use, the size of the circuit provides a good benchmark for evaluating extremely large circuits.

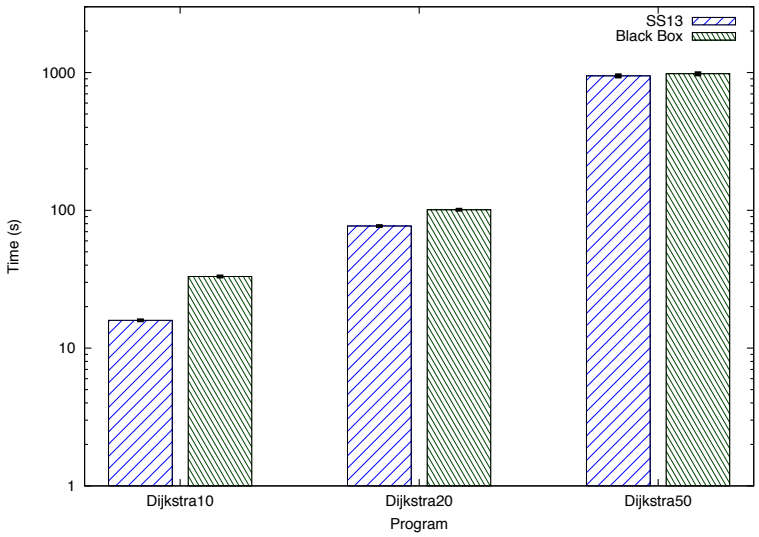


Figure 21: Dijkstra execution time in seconds. Note that for the largest input size, the execution overhead of outsourcing is almost non-existent.

5.6.2 Execution Time

With the mobile operations reduced to a minimal set, shown in Table 7, our experiments showed a diminishing cost of server overhead as the size of the test application increased. Considering Dijkstra's algorithm in Figure 21 shows that for a graph of 10 nodes, the outsourcing operations incur a 2.1x slowdown from running the protocol between two servers. However, as the number of graph nodes increases to 50, the confidence intervals for outsourced and server-only execution overlap, indicating a virtually non-existent overhead cost. When we compare these results to the gate counts shown in Table 6, we see that as the gate count for the underlying protocol increases, the additive cost of running the input MAC and output duplication amortize over the total execution time. This is to be expected from our predicted overhead of 15686 non-XOR gates for each CBC-MAC block in the input. However, since the mobile input for Dijkstra's algorithm is of a fixed size, we observe that increasing the application server input size does not add to the outsourcing overhead, showing the black box protocol to be more efficient for large circuit sizes with small mobile input.

When we consider a growing mobile input size, we observe the overhead cost of the MAC operation performed on the mobile input. In the matrix multiplication test program,

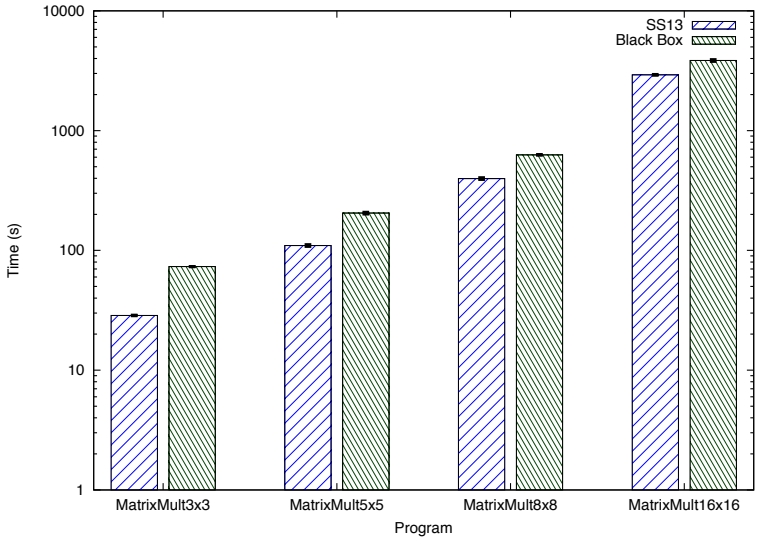


Figure 22: Matrix multiplication execution time in seconds. Note that the execution overhead still diminishes even as the mobile input size increases.

we observed a 2.6x slowdown for the smallest input size of a 3×3 matrix (Figure 22). As in the previous experiment, this overhead diminished to a 1.3x slowdown for the largest input size, but diminished at a slower rate when compared to the circuit size. This is a result of additional AES invocations to handle the increasing mobile input size. However, the reduction in overhead shows that even as input sizes increase, the circuit size is still the main factor in amortizing overhead.

In our final experiment, we considered a massive circuit representing one of the most complex garbled circuit programs evaluated to date. When comparing the outsourced execution to a standard two-party execution, the overhead incurred by the outsourcing operations is almost non-existent, as shown in Table 8. This experiment confirms the trends of diminishing overhead cost observed in the previous two experiments. From this and previous work, we know that evaluating large circuits from mobile devices is *not possible* without outsourcing the bulk of computation. Given that many real-world applications will require on the order of billions of gates to evaluate, this experiment shows that our black box outsourcing technique allows mobile devices to participate in secure two-party computation at roughly the same efficiency as two server-class machines executing the same computation.

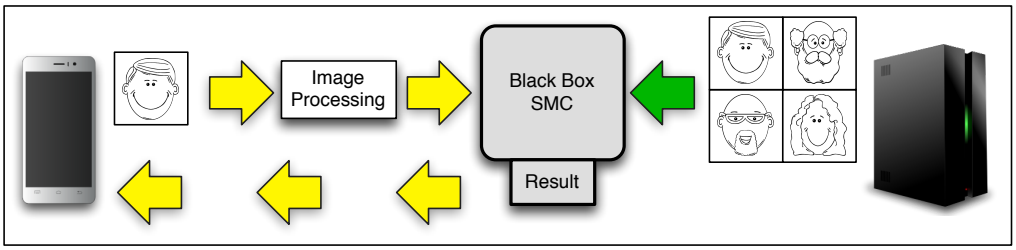


Figure 23: An example of the facial recognition application.

Table 8: Comparing SS13 and Black Box runtime. All times in seconds.

Program Name	SS13	BB	Increase
Dijkstra10	16 ± 1%	33 ± 1%	2.1x
Dijkstra20	77 ± 1%	100 ± 1%	1.3x
Dijkstra50	940 ± 2%	980 ± 2%	1.0x
MatrixMult3x3	28.6 ± 0.8%	73.2 ± 0.5%	2.6x
MatrixMult5x5	110 ± 2%	200 ± 2%	1.9x
MatrixMult8x8	400 ± 2%	627 ± 0.9%	1.6x
MatrixMult16x16	2900 ± 1%	3800 ± 2%	1.3x
RSA128	4700 ± 2%	4900 ± 3%	1.0x

5.6.3 Bandwidth

Because transmitting data from a mobile device is costly in terms of time and power usage, we attempted to minimize the amount of bandwidth required from the mobile device. Thus, the bandwidth used by the mobile device for any given application can be represented as a simple formula, shown in Table 7. Because this bandwidth is nearly minimal and easily calculated for any test program, we focused our experimentation on examining the bandwidth overhead incurred between the application server and the Cloud.

As in the case of execution time, Table 9 shows an inverse relation between circuit size and overhead cost. Before running the experiment, we predicted that the bandwidth overhead would approximately match the overhead in circuit size shown in Table 6. The experiments confirmed that the actual bandwidth overhead was equal to or slightly larger than the overhead in non-XOR gates in the circuit. The reason for this correlation is twofold. First, the free-XOR technique used in the shelat-Shen protocol allows XOR gates to be represented without sending any data over the network. Thus, adding additional XOR

Table 9: Comparing SS13 and Black Box bandwidth usage between the parties performing the generation and evaluation of the garbled circuit. All bandwidth in bytes.

Program Name	SS13	BB	Increase
Dijkstra10	2.44×10^9	3.87×10^9	1.6x
Dijkstra20	1.52×10^{10}	1.73×10^{10}	1.1x
Dijkstra50	2.02×10^{11}	2.05×10^{11}	1.0x
MatrixMult3x3	3.43×10^9	7.66×10^9	2.2x
MatrixMult5x5	1.57×10^{10}	2.56×10^{10}	1.6x
MatrixMult8x8	6.43×10^{10}	8.73×10^{10}	1.4x
MatrixMult16x16	5.11×10^{11}	6.01×10^{11}	1.2x
RSA128	8.69×10^{11}	8.72×10^{11}	1.0x

gates does not incur bandwidth cost. Second, in cases where the actual overhead is slightly larger than the circuit size overhead, we determined that the added cost was a result of additional oblivious transfers. These operations require the transmission of large algebraic group elements, so the test circuits which incurred increased overhead from the growth of the mobile input showed a slightly larger bandwidth overhead as well. Ultimately, as in the case of execution time, our experiments demonstrate that the black box outsourcing scheme incurs minimal bandwidth usage at the mobile device with diminishing bandwidth overhead between the application server and the Cloud.

5.7 Application: Facial Recognition

The growing number of mobile applications available present a wealth of potential for applying privacy-preserving computation techniques to the mobile platform. In Chapter 3, we demonstrated one potential application with their privacy-preserving navigation app. Mood et al. [127] presented a second app, a friend-finding application. Here we present a third mobile-specific application: facial recognition. In this setting, a secret operative or law enforcement agent carrying a mobile device needs to analyze a photo of a suspected criminal using an international crime database (see Figure 23). The database, managed by an international organization, would compare the photo to their database in a privacy-preserving manner, returning a match if the suspect appears in the database. In this scenario, the agent must keep the query data private to prevent insiders from learning who is being tracked,

Table 10: Runtime results for the privacy-preserving facial recognition application. Time indicates the total runtime of the garbled circuit part of the computation. All time in seconds.

Program Name	Time
FaceRec10	87.1 ± 0.9%
FaceRec100	170 ± 2%
FaceRec1000	1000 ± 2%

and the international organization must keep the database private from agents associated with any particular nation.

To implement this application, we use the facial recognition techniques developed for the Scifi protocol of Osadchy et al. [135]. They develop a technique for two servers to perform efficient facial recognition using discrete parameters, which can more easily be manipulated in secure computation protocols. They combine machine learning techniques in a preprocessing phase with a secure online phase that compares the hamming distance of photos represented as bit strings. To demonstrate our application, we implement the online comparison phase of this protocol in our black box outsourcing protocol (the $F_{threshold}$ function in their work). The mobile device provides a 928 bit representation of a photo, while the application server provides a database of representations containing 10, 100, and 1000 faces.

Our results show that given a database of 10 faces, the outsourced protocol can run the online phase in approximately 87 seconds (see Table 10). As the size of the facial database increases, the execution time for comparing across the entire database grows. This growing cost is a result of the large cost of representing the facial database as garbled input. Provided with a two-party SMC protocol that more efficiently computes over large data sets, our black box protocol could be used to move this application from feasible to practical. This demonstrates that an application designed and implemented to run between two servers can be feasibly executed from a mobile device. As new, more heavyweight applications are developed, our technique for outsourcing allows any of those applications to be executed from a mobile device with comparable efficiency to the server platform.

5.8 Comparison to Previous Techniques

While our implementation and evaluation in the previous section represents the first empirical analysis of black box outsourcing, two other protocols have been proposed in the literature, which we term KMR [92] and JNO [89]. We evaluate the tradeoffs between each technique in this section.

5.8.1 KMR

While the main focus of their work is the fixed outsourcing protocol Salus [92], Kamara et al. sketch a black box technique for outsourcing any two-party computation protocol. Essentially, their protocol encodes each bit of the Mobile input as a bit string of length k for some computational security parameter k. This encoded input, along with the mappings for reversing this encoding, is secret shared between Application and Cloud, and then restored using only XOR gates inside the circuit. A similar encoding technique is used to maintain both privacy and integrity of the output from the circuit. This technique has the advantage of adding only XOR gates to the circuit, which can be transmitted and evaluated cheaply using many SMC techniques. However, it also requires that the mobile input and output be expanded by a factor of k. By contrast, our evaluation demonstrates that the overhead caused by adding AND gates to the computation is minimal, and the Mobile bandwidth use is kept to $O(|x|)$ with a small constant multiple. This is particularly advantageous on smartphones, where data usage is often restricted by slow network speeds or provider-imposed bandwidth caps.

5.8.2 JNO

Developed concurrently to our protocol, Jakobsen et al. [89] presented a framework for outsourcing SMC protocols across any number of "worker" servers. Their protocol follows a similar procedure to our own, but they describe a novel MAC construction that allows the Mobile input to be checked using only inexpensive linear operations in the circuit. Essentially, their technique requires that the MAC key be committed at the start of the protocol, then opened once the rest of the input values are committed to the computation.

Once the key is opened, it can be multiplied as a known constant with the MOBILE input, which is secret shared according to the underlying SMC protocol. A simple multiplicative MAC can then be verified before computation continues. The advantage of this scheme is that it does not incur the k factor expansion of KMR while still adding only linear operations to the underlying circuit (e.g., XOR for boolean circuits). The tradeoff is that the underlying SMC protocol must allow for reactive computation (i.e., private values can be opened in the middle of computation). While this property is common in secret-sharing SMC protocols, it is difficult to achieve with garbled circuits. The generic technique for making garbled circuits into a reactive SMC protocol requires additional, MAC operations inside the circuit [76]. More efficient reactive garbled circuit protocols exist [58, 127], but require special constructions that cannot be combined with all garbled circuit protocols in a generic way. Our protocol allows for true black box outsourcing of *any* SMC protocol (reactive or non-reactive), and our empirical performance evaluation demonstrates that the overhead of adding AND operations to the circuit is minimal when the circuit size is large. This setting is preferable for computation that is more efficiently evaluated using garbled circuits than arithmetic secret-sharing SMC schemes.

5.9 Conclusion

While our previous outsourcing protocols both allow for significant performance improvements when executing garbled circuits on the mobile platform, it is unclear how these techniques can be applied to other SMC primitives that exist now or may be developed in the future. To resolve this limitation, this chapter presents a technique for outsourcing any two-party SMC protocol in a black box manner. Our black box protocol securely offloads the cost of the SMC protocol to the Cloud, providing maximal efficiency to the mobile device while maintaining strong security guarantees. Our performance evaluation shows that as the complexity of the program being evaluated increases, the cost of outsourcing diminishes. As a result, we enable execution of any SMC protocol from a mobile device at approximately the same efficiency as running the protocol between two servers.

CHAPTER VI

OUTSOURCING PRIVATE FUNCTION EVALUATION

6.1 Introduction

Our protocol for black box SMC outsourcing brings privacy-preserving computation to the mobile device with extremely high efficiency for the mobile device as well as diminishing overhead for the servers participating in the computation. Furthermore, as more efficient two-party SMC protocol are developed, along with more efficient circuit constructions for MAC primitives, the cost of our outsourced SMC will directly diminish as well. Given these results, the challenge of producing practical, deployable SMC on mobile devices now largely rests with solving the same problem on the desktop platform, which lies outside the scope of this work.

However, the security of outsourced SMC protocols can still be bolstered in a variety of real-world settings. Specifically, our black box protocol, as well as all of the existing protocols for outsourced SMC, require that all parties have knowledge of the application being executed. In a variety of real for government agencies and private companies, the computation itself may be confidential or proprietary information that should not be revealed to the mobile device or the Cloud. While it is possible to combine these techniques with universal circuits to achieve function privacy, universal circuit constructions incur a significant expansion in the size of the circuit being evaluated. For example, given a family of circuits of size g, Valiant's construction [157] for a universal circuit to evaluate this circuit family contains $19g \log g$ gates. Because circuit sizes are a significant limiting factor for all SMC techniques, several protocols for private function evaluation (PFE) have been developed in the non-outsourced setting with linear complexity with respect to the size of the circuit being evaluated. However, it is currently unknown how these protocols could be applied in the outsourced setting.

In this chapter, we develop the first linear outsourced PFE protocols secure against

semi-honest, covert, and malicious Cloud adversaries. These protocols allow an application server to execute a private function over inputs provided by a set of mobile devices that are aided by an untrusted Cloud server. To make these protocols possible, we developed a new technique for combining public garbled circuit computations with the privately evaluated function, which we term a "partially-circuit private" garbling technique.

Our setting and protocols constitute the following contributions:

- **Outsourced PFE Setting:** We develop a new setting for outsourcing private function evaluation against a variety of adversaries. We build our setting on the outsourced model for SMC developed in our previous work. However, we motivate a new trust model that allows for parties needing privacy guarantees to use any Cloud infrastructure without revealing any information about the computation being executed. Following this new motivation, we adapt the trust model used in these protocols to increase security against the Cloud.

- **Outsourced PFE Protocols:** Our primary contribution is a set of outsourced PFE protocols with security guarantees against semi-honest, covert, and malicious Cloud adversaries. To achieve this, we build on the garbled circuit-based PFE scheme of Mohassel and Sadeghian [126] and generalize their protocol according to the garbling scheme framework developed by Bellare et al. [16]. Given this technique for garbling circuits obliviously, we construct an outsourced PFE protocol that is secure against a semi-honest application server and both semi-honest and covert Cloud providers. Furthermore, we show that our covert secure protocol can also defend against malicious mobile parties. To achieve this, we leverage cut-and-choose techniques used in existing outsourced garbled circuit protocols [9] to ensure that all parties follow the protocol and learn nothing about the function being evaluated or the other inputs to the computation. Our proofs of security demonstrate that the Mohassel-Sadeghian garbling scheme, while originally proven secure against semi-honest adversaries, can be extended into stronger adversary models.

  While outsourced PFE is theoretically possible using an outsourced SMC protocol

executing an universal circuit, our protocols provide the first approach with linear complexity in the size of the private function. Furthermore, our protocols require the mobile parties to perform operations that are only depended on the input and output length. These operations can be implemented in practice using fast, symmetric-key operations.

- **PCP Garbling:** Our outsourced PFE protocol for a covert Cloud adversary provides efficient security for a variety of applications. However, to extend our results to all possible Cloud adversaries, we construct a third outsourced PFE protocol that is secure against a malicious Cloud and malicious mobile devices. Our chief technical innovation to make this protocol possible is the development of a partially-circuit private (PCP) garbling scheme, a technique which allows us to stitch together public and private functions in a single computation by alternating garbling techniques. This allows preprocessing input checks and post-processing output preparation to be ensured by all parties rather than just the party providing the private function as input. In this way, we can apply input consistency checks and output encryption via one-time pad using auxiliary circuits as in previous SMC protocols [150, 124, 151, 109]. While we apply this technique expressly for the purpose of adding consistency checks, the garbling technique stands as an independent contribution, as it is useful in practice to provide arbitrary public preprocessing and post-processing for any garbled circuit-based PFE protocol.

6.2 Setting and Background

Our protocols combine a range of underlying SMC techniques for circuit garbling and for ensuring that adversarial behavior is caught during protocol execution. Here we provide a summary of our modified outsourced setting and the underlying constructions use in our protocols. We refer the reader to the original work for further discussion.

6.2.1 Outsourced PFE Setting

Our outsourced PFE setting is modeled closely after the outsourced SMC setting used in the previous chapters. However, the trust model in the PFE setting requires a more nuanced approach, and the motivation for why this trust model fits is slightly different from our previous work. Here we define the parties in our outsourced PFE protocols and the trust model for each.

Application: We define this party as the application server that is hosting the application and providing the private function to be evaluated. Our setting assumes that this party will always provide the function and may or may not provide input and receive output from the computation. In addition, we assume this party to be semi-honest for all protocols.
Mobile: We define this party (these parties) as one (or more) computationally-restricted devices that provide input to the application server and may receive some output from the computation. We develop protocols that are secure in the presence of semi-honest and malicious mobile devices. While many applications can also model the mobile device as a semi-honest adversary, it is possible that maintaining the privacy of the evaluated function from the mobile participant is critical to the application.
Cloud: Our final participant is the Cloud server that assists the mobile device(s) in executing the costly operations in our PFE protocols. Since mobile users may be accessing variable Cloud infrastructure in any particular application scenario, we seek to develop protocols that remain secure against any adversary (semi-honest, covert, and malicious).

There are two reasons why we model the function holder as a semi-honest adversary. First, recent work in the general PFE space has produced protocols to defend against malicious function holders [125]. However, the practicality of such a guarantee is not universally applicable. If the function holder receives output from the computation, even a protocol that is secure against a malicious function holder cannot prevent that party from providing a function that reveals other parties' inputs.

Second, this setting accurately models a large number of real-world applications. A

plethora of smartphone applications are built with the architecture of a client running on the mobile device contacting an application server for information and data processing. While the user may trust the application to provide useful and correct functionality, he may wish to limit the application server's access to his data in the event that the application server is later compromised. Given these two parties that can be trusted to follow a PFE protocol, it is reasonable to model the Cloud as the strongest possible adversary while modeling the application server as semi-honest. This allows the mobile device to use *any* publicly available computing infrastructure to assist with the computation, from high-profile Cloud services to public access library terminals. For these reasons, as the first step in this space we prioritize security against the Cloud, and add guarantees against other adversaries as a secondary goal.

6.2.2 Garbling Scheme Definitions

To allow for a general approach to proving security in garbled circuit protocols, Bellare et al. [16] developed a cryptographic definition for a garbling scheme as well as several notions of security that are necessary to provide security in SMC protocols. They define a garbling scheme as a five-tuple $\mathcal{G} = (Gb, En, De, Ev, ev)$. Given the description f of a function that we wish to compute securely, the function $ev(f, \cdot) : \{0,1\}^n \rightarrow \{0,1\}^m$ executes f. $Gb(1^k, f) \rightarrow (F, e, d)$ is a probabilistic function that takes a function and security parameter as input and produces a garbled representation F, and the strings e and d which are used to describe the encode function $En(e, \cdot)$ and decode function $De(d, \cdot)$. These functions are used to encode the input to the garbled circuit, and decode the output after the garbled circuit has been evaluated. Finally, $Ev(F, \cdot)$ evaluates the garbled circuit with an encoded input. Correctness for the scheme is guaranteed by the requirement that for any $x \in \{0,1\}^{f.n}$ and $(F, e, d) \in [Gb(1^k, f)]$, then $ev(f, x) = De(d, Ev(F, En(e, x)))$.

Many practical garbled circuit protocols also require that a garbling scheme have a projective property. Bellare et al. define this property as follows: if, for all $x, x' \in \{0,1\}^{f.n}, k \in \mathbb{N}$, and $i \in [1, ..., n]$, when $(F, e, d) \in [Gb(1^k, f)], X' = En(e, x')$, and $X = En(e, x)$, then $X = (X_1, ..., X_n)$ and $X' = (X'_1, ..., X'_n)$ are n vectors, $|X_i| = |X'_i|$, and $X_i = X'_i$ if x and x'

have the same i^{th} bit. For our partially-circuit private garbling scheme, all garbling schemes used must be projective garbling schemes.

In addition, Bellare et al. define a side-information function $\Phi(f)$ that defines what information is revealed about the function f by the garbled representation F. For the purpose of our protocols, we specify two specific side functions from their work. For a function f, $\Phi_{size}(f) = (n, m, q)$ reveals the size of the circuit as having n input bits, m output bits, and q gates. If the circuit does not hide any information about the function, we say that $\Phi_{circ}(f) = f$.

Finally, Bellare et al. develop five different definitions of security for garbled circuits, providing three distinct guarantees: privacy (when the evaluating party receives output from the function), obliviousness (when the evaluating party does not receive output), and authenticity (to ensure that the evaluating party cannot tamper with the results of the computation). Our protocols require a garbling scheme to provide privacy under the prv.sim definition of privacy. This game-based definition specifies a game PrvSim that is given a garbling scheme, an adversary, a side-information function, and a security parameter. The adversary is allowed a single garbling query for a function f and input x of its choosing. The adversary is returned the tuple (F, X, d), where this tuple is, with uniform probability, either prepared according to the real garbling scheme $\mathcal{G}$, or is produced by a simulator $\mathcal{S}(1^k, y, \Phi(f))$ that only has access to the output $y \leftarrow ev(f, x)$ and the side-information function $\Phi(f)$. A garbling scheme is said to be prv.sim secure over a side information function $\Phi(f)$ if for every polynomial time adversary $\mathcal{B}$, there is a polynomial time simulator $\mathcal{S}$ such that the advantage of the adversary:

$$Adv_{\mathcal{G}}^{prv.sim,\Phi,\mathcal{S}}(\mathcal{B}, k) = 2Pr[PrvSim_{\mathcal{G},\Phi,\mathcal{S}}^{\mathcal{B}}(k)] - 1$$

is negligible.

6.2.3 PFE with Garbled Circuits

Mohassel and Sadeghian [126] define a framework for general PFE by breaking the protocol into two generic steps: hiding circuit topology and evaluating a single gate in a private

manner. Using this framework, they develop a two-party, constant-round protocol for performing both steps using Yao's garbled circuits. Their construction requires a linear number of operations with respect to the size of the circuit being evaluated and is secure against semi-honest adversaries. Their protocol specifies the circuit generator as the party providing input, while the circuit evaluator provides the circuit to be computed and optionally an additional input value. To maintain privacy of the circuit topology, their protocol begins with the generating party defining random wire labels for every wire in the circuit. Then, using a construction termed the oblivious extended permutation (OEP), the evaluating party defines which gate output wires are connected as inputs to subsequent gates using blinding values to prevent the generating party from learning these mappings. Once the generating party receives the blinded input wires for each gate, he garbles the gates and returns the garbled circuit to the evaluator. The evaluator then evaluates the circuit following Yao's protocol by applying the blinding value at each gate to recover the correct output wire value. Rather than developing a special protocol for private gate evaluation, they simply define all functions using NAND gates, so that the generator cannot infer any information from the functionality of the gates.

To implement the OEP functionality, they define two constructions. The first uses an oblivious switching network (OSN) to blind and permute the wire values in an offline/online protocol. The second construction uses partially homomorphic encryption to allow the evaluator to add in blinding values for each input wire while not learning the actual garbled wire value. We build our protocol on the simpler construction using homomorphic encryption. As an additional contribution, we define their protocol in Section 6.3 and prove that it is prv.sim secure in Section 6.4 according to the definition of Bellare et al. [16].

Intuitively, the combination of the OEP functionality and the use of NAND gates prevents the generator from learning anything about the circuit beyond the size and the number of input and output bits. The use of the homomorphic blinding or OSN prevents the evaluator from learning the garbled wire values, which preserves the privacy of the generator's inputs based on the guarantees of the garbled circuit construction. However, given a malicious generator, the security of the scheme does not hold. To protect against a malicious or

covert adversary, a PFE protocol must handle the same malicious scenarios as any garbled circuit SMC protocol (i.e., ensure that the circuit is garbled consistently and that any oblivious transfers deliver correct input wire labels). In addition, we must ensure that a malicious generator cannot learn anything about the function during the partially homomorphic OEP protocol.

6.2.4 Outsourcing SMC Techniques

To develop an outsourced PFE protocol, we examine techniques from previous outsourced SMC protocols. However, many of these techniques do not directly apply to the PFE setting. For example, recent protocol developments have shown that any SMC protocol can be transformed into an outsourced protocol using black box techniques as in Jakobsen et al. [89] or our protocol in Chapter 5. These protocols work by adding security checks into the evaluated circuit itself. In the PFE setting, such a protocol must *always* assume an honest function holder. If we hope to improve the security to defend against a malicious function holder, we cannot rely on black box techniques that provide security checks inside the private function.

Our approach uses techniques from Kamara et al. [92] and the protocol described in Chapter 4 to allow the mobile participants to jointly generate the randomness used to garble the circuit. This randomness is then passed to the Cloud for garbling a set of gates that can be assembled and checked using a cut-and-choose that is modified for the PFE setting to ensure the circuit is constructed correctly and that no information about the function is leaked to the generator. As a secondary contribution, these security checks can be applied to the original two-party protocol of Mohassel and Sadeghian to provide security guarantees against a covert or malicious generator.

6.2.5 Security Definition

To capture the added security guarantee of function privacy in the outsourced setting, we slightly modify the definition for secure outsourced SMC used in Chapter 3. As in our previous work, we demonstrate security by showing that the real world protocol execution can be simulated in an ideal setting with a trusted third party. However, rather than

providing the function as a public parameter to all parties, we allow the simulator controlling the Application server to include the circuit as a private input to the trusted third party in the ideal world. By treating the circuit as an input to the computation, the input privacy captured by the existing SMC definition encompasses the function privacy that we wish to achieve in our PFE setting.

6.2.6 Notation

Here we define the notation used in the following sections. We describe the circuit hiding garbling protocol of Mohassel and Sadeghian [126] (which we later refer to as MS13) using a slight modification on the general notation of Bellare et al. [16] as $\mathcal{G} = (Gb_{MS}, En_{MS}, De_{MS}, Ev_{MS}, ev)$. The function $Gb_{MS}(1^k, f, r) \rightarrow (F, e, d)$ uses the random seed r to output a garbled circuit F using that is composed of two components (G, B). G is the garbled representation of the circuit, and is delivered to the generator, while B is a set of blinding values maintained by the evaluator during the OEP protocol. One critical note for the security of our protocol is that G reveals no information about the topology of the underlying circuit without possession of B. Finally, we also assume a projective garbling scheme $Gb(1^k, f, r)$ to be a Yao-based circuit garbling technique, with inputs defined as previously stated.

6.3 Private Function Garbling

For reference, we define the semi-honest PFE scheme developed by Mohassel and Sadeghian [126] in the generalized garbled circuit notation developed by Bellare et al. [16]. For ease of comparison to our work, we refer to the garbling party as Cloud and the evaluating party as Application. In the following section, we provide a new proof that this garbling scheme satisfies prv.sim security.

Given the functions in Figure 24 and Figure 25, the remaining functions in the garbling scheme En, De, Ev, ev are trivially realized. En and De simply consist of mapping a real bit value b to a garbled wire value w^b and vice versa. Ev corresponds to the standard Yao evaluation protocol, with the exception that Application adds the appropriate blind (based on the permutation vector v') back to the previous output key before evaluating the

Common inputs: a computational security parameter k and the side information function $\Phi_{size}(C) = (n, m, q)$ for APPLICATION's private function C. Here, n is the number of input wires, m is the number of output wires, and q is the number of gates in C.
Private inputs: APPLICATION inputs a circuit C that corresponds to her private function. CLOUD inputs a random seed r.
Outputs: APPLICATION receives the garbled function G, a vector of blinding values B, and a vector d that describes the decoding function $De(d, \cdot)$ that recovers the final output from the garbled output.

1. Using r to seed a pseudorandom generator, CLOUD generates $n+q$ sets of topologically ordered wire labels w_i^0, w_i^1, where the indices $i \in [1, ..., n]$ correspond to input wires, $i \in [n+1, ..., n+q-m]$ correspond to gate output wires that are input to other gates, and $i \in [n+q-m+1, ..., n+q]$, and circuit output wires.

2. CLOUD generates a random permutation string v from r where $|v| = n + q$ and permutes the wire labels as $w_i^{v_i}, w_i^{\overline{v}_i}$. He then appends each permutation bit to the wire label as $w_i^{v_i}||v_i, w_i^{\overline{v}_i}||v_i$.

3. CLOUD inputs $w_i^{v_i}||v_i, w_i^{\overline{v}_i}||v_i$ for $i \in [1, ..., n+q-m]$ into the $CTH(\cdot)$ functionality, while APPLICATION inputs the circuit C. CLOUD receives a vector of blinded input wires bw_j^0, bw_j^1 for $j \in [1, ..., 2q]$ that are topologically ordered input wires for the gates of C.

4. Given the ordered and blinded gate input wires bw_j^0, bw_j^1 for $j \in [1, ..., 2q]$ and the ordered gate output wires w_i^0, w_i^1 for $i \in [n+1, ..., n+q]$, CLOUD garbles each gate following Yao's garbling protocol and using the NAND functionality for all gates.

5. CLOUD sends the garbled gates G and the hash of the output wire mappings $d = (H(w_i^0), H(w_i^1))$ for $i \in [n+q-m+1, ..., n+q]$ to APPLICATION. The vector of blinding values B is retained by APPLICATION from the $CTH(\cdot)$ functionality.

Figure 24: The private function garbling protocol by Mohassel and Sadeghian

next gate. ev is simply the function f that corresponds to the circuit C.

6.4 Proof of MS13

Here we prove the prv.sim security of Mohassel and Sadeghian's garbling scheme.

Theorem 1. *The scheme Gb_{MS} is prv.sim secure over the size-information function Φ_{circ}.*

We set an adversary A such that the tuple of garbled circuit, garbled input, and decoding function (F, X, d) is indistinguishable between the real protocol and a simulated view where the simulator is given $\Phi_{circ}(f)$ and $f(x)$. To do this, we build a simulator S through the following series of hybrid experiments. We slightly modify the notation of Bellare et

Private inputs: APPLICATION inputs a circuit C that corresponds to her private function. CLOUD inputs a set of permuted wire labels $w_i^{v_i}||v_i, w_i^{\overline{v_i}}||v_i$ for $i \in [1, ..., n+q-m]$.
Outputs: APPLICATION receives a vector of blinding values B and CLOUD receives a vector of blinded input wires bw_j^0, bw_j^1 for $j \in [1, ..., 2q]$ that are topologically ordered input wires for the gates of C.

1. CLOUD encrypts each wire label using his public key pk for a semantically secure, additive-homomorphic public key scheme, resulting in pairs $E_{pk}(w_i^{v_i}||v_i), E_{pk}(w_i^{\overline{v_i}}||v_i)$. CLOUD sends the permuted and encrypted labels to APPLICATION.

2. APPLICATION generates pairs of random blinding values b_j^0, b_j^1 for $j \in [1, ..., 2q]$ to blind the input labels for each gate. In addition, she generates a second permutation string v' where $|v'| = 2q$. For the ith gate in topological order, with input wires j and k, APPLICATION encrypts

$$E_{pk}(b_{2j-1}^{v'_{2j-1}}||v'_{2j-1}), E_{pk}(b_{2j-1}^{\overline{v'_{2j-1}}}||v_{2j-1}), E_{pk}(b_{2j}^{v'_{2j}}||v'_{2j}), E_{pk}(b_{2j}^{\overline{v'_{2j}}}||v_{2j})$$

3. APPLICATION then homomorphically adds the blind to the permuted keys by using the inverse permutation function to find $\pi^{-1}(2j) = k$ for $j \in [1, ..., 2q]$

$$E_{pk}(b_{2j-1}^{v'_{2j-1}}||v'_{2j-1}), E_{pk}(b_{2j-1}^{\overline{v'_{2j-1}}}||v_{2j-1}) + E_{pk}(w_{k-1}^{v_{k-1}}||v_{k-1}), E_{pk}(w_{\overline{k-1}}^{\overline{k-1}}||v_{k-1}),$$
$$E_{pk}(b_{2j}^{v'_{2j}}||v'_{2j}), E_{pk}(b_{2j}^{\overline{v'_{2j}}}||v'_{2j}) + E_{pk}(w_k^{v_k}||v_k), E_{pk}(w_{\overline{k}}^{v_{\overline{k}}}||v_{\overline{k}})$$

   APPLICATION then returns the resulting ciphertexts to CLOUD in topological ordering.

4. CLOUD decrypts the input wire labels, and for the ith gate in topological order, recovers the correct blinded input wire label by permuting based on the appended bit, returning the labels to their correct ordering (because, for the bit b, $b \oplus (v_j \oplus v'_{2j-1}) \oplus (v_j \oplus v'_{2j-1}) = b$).

Figure 25: The $CTH(\cdot)$ functionality by Mohassel and Sadeghian

al. [16] and notate the prv.sim game as $PrvSim^A_{MS,\Phi_{circ},S(k,b)}$, where $b = 1$ denotes the real protocol and $b = 0$ denotes the simulated view. Note that for this garbling scheme, the garbled function $F = (G, B)$, where G is the set of garbled gates and B is the set of blinding values generated during garbling.

$Hyb1^A_{MS,\Phi_{circ},S(k)}$: This experiment is identical to $PrvSim^A_{MS,\Phi_{circ},S(k,1)}$ except that the experiment garbles the output wires of G to produce the fixed output $y = f(x)$.

Lemma 45. $PrvSim^A_{MS,\Phi_{circ},S(k,1)} \overset{c}{\approx} Hyb1^A_{MS,\Phi_{circ},S(k)}$

Proof. This follows from the original proof of Yao's protocol. Since the adversary is only given a single garbled input X, they can only decrypt a single entry at each garbled gate,

guaranteed by the semantic security of the underlying encryption scheme. Because they cannot decrypt any *other* output wire labels, a real circuit garbling and a fixed output garbling are indistinguishable. □

$Hyb2^{A}_{MS,\Phi_{circ},S(k)}$: This experiment is identical to $Hyb1^{A}_{MS,\Phi_{circ},S(k)}$ except that the experiment provides a random garbled input X' instead of the real garbled input X.

Lemma 46. $Hyb1^{A}_{MS,\Phi_{circ},S(k)} \overset{c}{\approx} Hyb2^{A}_{MS,\Phi_{circ},S(k)}$

Proof. This follows from the garbling function $En_{MS}(e,\cdot)$, which produces a uniformly random string to represent each garbled input bit. Since the distribution of these strings is indistinguishable for any input x, then the garbled inputs $X = En_{MS}(e,x)$ and $X' = En_{MS}(e,x')$ are indistinguishable. □

Lemma 47. $Hyb2^{A}_{MS,\Phi_{circ},S(k)}$ *runs in polynomial time.*

Proof. This lemma follows trivially since the real world garbling protocol runs in polynomial time and each intermediate hybrid adds a constant number of operations. □

We conclude the proof by noting that $Hyb2^{A}_{MS,\Phi_{circ},S(k)} \equiv PrvSim^{A}_{MS,\Phi_{circ},S(k,0)}$. S follows the real world protocol where the hybrid experiment does not differ, garbling the function f given $\Phi_{circ}(f)$ and $y = f(x)$ (with the exception of the output wire modification in Lemma 45). From Lemma 45-47, S proves Theorem 1.

6.5 Semi-Honest Outsourced PFE

To illustrate our outsourcing techniques in a simplified setting, we first define a protocol for outsourcing PFE in the presence of semi-honest adversaries. Not only does this protocol serve as a warm up to stronger security settings, it could also be applied to applications requiring higher efficiency with less need for strong assurance.

6.5.1 Protocol Overview

At a high level, our outsourcing protocol (described in detail in Figure 26) essentially splits the role of the circuit generator across all of the mobile devices participating in the computation. These devices must initiate the computation with a shared secret that will

then be used by the Cloud as the randomness to generate the garbled wire values. This shared secret can be generated using a secure coin flip or a key exchange protocol. Next, the Cloud and the Application server engage in the MS13 circuit hiding garbling process to prepare the function for evaluation. Then, since each mobile device possesses the random seed used to garble the circuit, it can generate the garbled wire labels that correspond to its input and directly deliver their input to the Application server for evaluation. Finally, the Application server delivers the garbled output to each party and receives a table to ungarble her output wires (if she receives any output from the computation). Again, using the shared random seed, the mobile parties can then generate their own ungarbling table and recover their output.

Note that unlike other outsourced SMC protocols, this protocol does not achieve any sort of fairness, as the Application server receives her output before all other parties. However, fairness can be incorporated into this protocol by having CLOUD blind all of the output values with one-time pads, which it can release simultaneously at the end of the protocol.

6.5.2 Security

The security of our scheme essentially reduces to the underlying PFE scheme. CLOUD learns nothing about the private function based on the topology-hiding property of Mohassel and Sadeghian's protocol, while APPLICATION learns nothing about any party's input based on the privacy of Yao's garbled circuit protocol. Finally, since CLOUD never observes any MOBILE input in any form and receives no output from the circuit, it cannot learn anything about the inputs to the computation. We provide a complete proof of security in Section 6.6.

6.5.3 Complexity

Our protocol essentially matches the complexity of Mohassel and Sadeghian for the Application server and Cloud, with the only computation for the mobile devices being the input and output preparation and recovery. For the mobile devices input m_i and output y_i, this amounts to $O(|m_i| + |y_i|)$ symmetric key operations. While Mohassel and Sadeghian provide a more thorough discussion of the complexity of their protocol, it essentially requires that the Cloud generate $O(g)$ wire values, encrypt these, decrypt the result, and garble the

Common inputs: a computational security parameter k and the side information function $\Phi_{size}(C) = (n, m, q)$ for APPLICATION's private function C. Here, n is the length of a single party's input, m is the length of a single party's output, and q is the number of gates in C. All MOBILE participants share a common random seed s where $|s| = k$.
Private inputs: For every $i \in [2, ..., N]$, MOBILE$_i$ inputs a string m^i. APPLICATION inputs a bit string a and a circuit C that evaluates her private function $f(a, m^2, ..., m^N)$.
Outputs: APPLICATION receives the output y_1 and MOBILE$_i$ receives the output y_i for $i \in [2, ..., N]$.

1. The MOBILE devices deliver s to CLOUD.
2. CLOUD and APPLICATION run the interactive protocol $Gb_{MS}(1^k, C, s) \rightarrow (G, B, e, d)$, with CLOUD as generator using the seed s and APPLICATION as the evaluator with the private circuit C. After the protocol, CLOUD receives the garbled circuit G and APPLICATION receives a vector of blinding values B. CLOUD delivers the garbled circuit G to APPLICATION along with output decoding function d for APPLICATION's output wires only.
3. APPLICATION and CLOUD run k instances of an oblivious transfer protocol (using OT-extensions). After the protocol, APPLICATION receives her garbled input $En_{MS}(e, a)$.
4. Every MOBILE participant delivers his garbled input $En_{MS}(e, m_i)$ to APPLICATION.
5. APPLICATION runs $Ev_{MS}([G, B], [En_{MS}(e, a), En_{MS}(e, m_2), ..., En_{MS}(e, m_N)])$ to evaluate the garbled circuit, producing a vector of output values $[Y_1, ..., Y_N]$.
6. APPLICATION recovers her output as $De_{MS}(d, Y_1) = y_1$. She then delivers the garbled output wires Y_i to the ith MOBILE participant for $i \in [2, ..., N]$.
7. The ith MOBILE participant recovers his output as $De_{MS}(e, Y_i) = y_i$.

Figure 26: Semi-honest Outsourced PFE Protocol

gates, yielding a complexity of $O(g)$ symmetric key operations and $O(g)$ asymmetric key encryptions for the additive homomorphic encryption. The Application server must then generate $O(g)$ blinding values, encrypt these, add them homomorphically to the wire values sent by the generator, then evaluate the circuit using these blinding values. This yields $O(g)$ symmetric key operations, $O(g)$ asymmetric key encryptions, and $O(g)$ homomorphic additions. Finally, using OT extensions, the Cloud and Application server must also perform $O(k)$ public key operations to garble the Application server's input. For a summary of each party's complexity, see Table 11.

Table 11: The computational complexity of our semi-honest outsourced PFE protocol. Note that HE signifies additively homomorphic encryptions, HA signifies homomorphic addition.

Mobile	$O(\|m_i\| + \|y_i\|)$
Application server	$O(g) + O(g \text{ HE}) + O(g \text{ HA}) + O(k)$
Cloud	$O(g) + O(g \text{ HE}) + O(k)$

6.6 Semi-honest Protocol Proof

Here we prove the security of our outsourced PFE protocol against semi-honest adversaries according to the following theorem.

Theorem 2. *The outsourced PFE protocol securely and privately computes a circuit C when the Application server is semi-honest and non-colluding, the Cloud is semi-honest and non-colluding, and the mobile devices are semi-honest.*

6.6.1 Semi-honest Mobile

Here we construct a simulator S_M that can simulate the view of a semi-honest MOBILE adversary M^*. Without loss of generality, this party represents any number of colluding MOBILE devices executing the protocol. We construct this simulator through a set of hybrid experiments.

$Hyb1^{(M)}(k, x, f; r)$: This experiment is identical to $REAL^{(M)}(k, x, f; r)$ except that the experiment sends M^*s input m to the trusted third party instead of entering it into the real world protocol.

Lemma 48. $REAL^{(M)}(k, x, f; r) \stackrel{c}{\approx} Hyb1^{(M)}(k, x, f; r)$

Proof. Since all parties are behaving semi-honestly, the experiment can send the input it used to run M^* to the trusted third party. The trusted party will then return the result of computation $y = f(x)$ to the experiment, which will be identical to the result output by the circuit executed in the real world. □

$Hyb2^{(M)}(k, x, f; r)$: This experiment is identical to $Hyb1^{(M)}(k, x, f; r)$ except that the experiment uses s, received from M^* at the beginning of the protocol, to generate the output

wire labels used in the real world protocol. It then garbles the value y received from the trusted third party and returns go_{M^*} to M^*

Lemma 49. $Hyb1^{(M)}(k, x, f; r) \stackrel{c}{\approx} Hyb2^{(M)}(k, x, f; r)$

Proof. This hybrid holds since the garbled wire values in the real world protocol are generated deterministically based on s. In addition, since all parties are semi-honest, the resulting output y will be identical in both experiments. □

Lemma 50. $Hyb2^{(M)}(k, x, f; r)$ *runs in polynomial time.*

Proof. This lemma follows trivially since a real world execution of the protocol runs in polynomial time and each intermediate hybrid adds only constant time operations. □

We conclude the proof by letting S_M execute $Hyb2^{(M)}(k, x, f; r)$, following the real world protocol where the hybrid experiment does not differ. S_M runs M^* and controls Cloud and Application. S_M simulates the real world protocol and outputs whatever M^* outputs at the end of the simulation. From Lemma 48-50, S_M proves Theorem 2 when the Mobile parties are semi-honest.

6.6.2 Semi-honest Application

Here we construct a simulator S_A that can simulate the view of a semi-honest Application adversary A^*. We construct this simulator through a set of hybrid experiments.

$Hyb1^{(A)}(k, x, f; r)$: This experiment is identical to $REAL^{(A)}(k, x, f; r)$ except that the experiment sends A^*s inputs $f(\cdot), x$ to the trusted third party.

Lemma 51. $REAL^{(A)}(k, x, f; r) \stackrel{c}{\approx} Hyb1^{(A)}(k, x, f; r)$

Proof. Since A^* is semi-honest, the experiment invokes the trusted party using the input function $f(\cdot)$ and the input data x that it provided to A^* at the start of the experiment. The value returned by the trusted third party $y = f(x)$ will be identical to the value output by the circuit evaluated in the real world. □

$Hyb2^{(A)}(k, x, f; r)$: This experiment is identical to $Hyb1^{(A)}(k, x, f; r)$ except that during the $Garble_{CTH}()$ execution, the experiment runs the simulator S_{MS} to simulate A^*s view of the garbled circuit.

Lemma 52. $Hyb1^{(A)}(k, x, f; r) \stackrel{c}{\approx} Hyb2^{(A)}(k, x, f; r)$

Proof. Since the experiment knows topologically which gates feed into output wires of the private function $f(\cdot)$, it can garble those gates to always output the same value y. Based on Theorem 1, we know that S_{MS} exists and can indistinguishably simulate the view of the garbling scheme. □

Lemma 53. $Hyb2^{(A)}(k, x, f; r)$ *runs in polynomial time.*

Proof. This lemma follows trivially since the experiment runs the real world protocol and the simulator S_{MS}, both of which run in polynomial time. □

We conclude the proof by letting S_A execute $Hyb2^{(A)}(k, x, f; r)$, following the real world protocol where the hybrid experiment does not differ. S_A runs A^* and controls Cloud and Mobile. S_A simulates the real world protocol and outputs whatever A^* outputs at the end of the simulation. From Lemma 51-53, S_A proves Theorem 2 when the Application party is semi-honest.

6.6.3 Semi-honest Cloud

Here we construct a simulator S_C that can simulate the view of a semi-honest Cloud adversary C^*. We construct this simulator through a set of hybrid experiments.

$Hyb1^{(C)}(k, x, f; r)$: This experiment is identical to $REAL^{(C)}(k, x, f; r)$ except that the experiment garbles a random function $f'(\cdot)$ during the $Gb_{MS}()$ invocation instead of the function $f(x)$ used in the real world invocation, such that $\Phi_{size}(f') = \Phi_{size}(f)$

Lemma 54. $REAL^{(C)}(k, x, f; r) \stackrel{c}{\approx} Hyb1^{(C)}(k, x, f; r)$

Proof. Based on the proof of security from Mohassel and Sadeghian [126], a simulator S_{CTH} exists that can simulate the view of C^* given only $\Phi_{size}(f)$. □

$Hyb2^{(C)}(k, x, f; r)$: This experiment is identical to $Hyb1^{(C)}(k, x, f; r)$ except that during the oblivious transfer execution, the experiment provides a random input x' instead of APPLICATION's real input x.

Lemma 55. $Hyb1^{(C)}(k, x, f; r) \overset{c}{\approx} Hyb2^{(C)}(k, x, f; r)$

Proof. Based on the security guarantees of the OT protocol used, a simulator S_{OT} can simulate C^* view of a real protocol invocation using a random input x'. □

Lemma 56. $Hyb2^{(C)}(k, x, f; r)$ *runs in polynomial time.*

Proof. This lemma follows trivially since the experiment runs the real world protocol with two polynomial time simulators. □

We conclude the proof by letting S_C execute $Hyb2^{(C)}(k, x, f; r)$, following the real world protocol where the hybrid experiment does not differ. S_C runs C^* and controls APPLICATION and MOBILE. S_C simulates the real world protocol and outputs whatever C^* outputs at the end of the simulation. From Lemma 54-56, S_C proves Theorem 2 when the CLOUD party is semi-honest.

6.7 Covert Server Outsourced PFE

While the semi-honest secure protocol in the previous section provides a foundation for outsourcing PFE, it does not capture the necessary security guarantees for many realistic applications. Our original motivation for developing protocols in this setting is to allow users to participate in secure computation with any available Cloud resource, trusted or untrusted. Because many Cloud providers wish to maintain a reputation as reliable and trustworthy, they may not cheat if there is some possibility of being caught. This behavior is modeled by the covert security model, allowing a balance between efficiency and security guarantees. To meet this application scenario, we developed an outsourced PFE protocol that is secure against a covert Cloud server and malicious mobile devices.

Common inputs: a computational security parameter k, a statistical security parameter λ, and the side information function $\Phi_{size}(C) = (n, m, q)$ for APPLICATION's private function C. Here, n is the length of a single party's input, m is the length of a single party's output, and q is the number of gates in C. All MOBILE participants share a common random seed s where $|s| = k$.
Private inputs: For every $i \in [2, ..., N]$, MOBILE$_i$ inputs a string m^i. APPLICATION inputs a bit string a and a circuit C that evaluates her private function $f(a, m^2, ..., m^N)$.
Outputs: APPLICATION receives the output y_1 and MOBILE$_i$ receives the output y_i for $i \in [2, ..., N]$.

1. The MOBILE devices deliver s to CLOUD.
2. CLOUD ensures that every copy of s received is identical. If any of the values differs, CLOUD aborts. Otherwise, CLOUD uses a pseudorandom generator seeded with s to produce random seeds r_i for $i \in [1, ..., \lambda]$.
3. For every $i \in [1, ..., \lambda]$, CLOUD and APPLICATION run the interactive protocol $Gb_{MS}(1^k, C, r_i) \rightarrow (G_i, B_i, e_i, d_i)$, with CLOUD as generator using the seed r_i and APPLICATION as the evaluator with the private circuit C. After the protocol, CLOUD receives the garbled circuit G_i and APPLICATION receives a vector of blinding values B_i. CLOUD delivers all garbled circuits to APPLICATION along with output decoding functions d_i for APPLICATION's output wires.
4. For every $i \in [1, ..., \lambda]$, APPLICATION and CLOUD run k instances of a covert-secure oblivious transfer protocol (using OT-extensions). After the protocol, APPLICATION receives her garbled input $En_{MS}(e_i, a)$.
5. APPLICATION selects a random and uniform index $E \in [1, ..., \lambda]$. For every $i \in [1, ..., \lambda] \setminus E$, CLOUD sends r_i to APPLICATION. APPLICATION runs $Gb_{MS}(1^k, C, r_i) \rightarrow (G'_i, B'_i, e'_i, d'_i)$ and checks that:
   (a) $G_i = G'_i$
   (b) $d_i = d'_i$
   (c) $En(e_i, a) = En(e'_i, a)$

   If any of these checks fails, APPLICATION aborts.
6. Every MOBILE participant $j \in [1, ..., N]$ delivers his garbled input $En_{MS}(e_E, m_j)$ to APPLICATION.
7. APPLICATION runs $Ev_{MS}(G_E, B_E, En_{MS}(e_E, a), En_{MS}(e_E, m_2), ..., En_{MS}(e_E, m_N))$ to evaluate the selected evaluation circuit, producing output values $[Y_1, ..., Y_N]$.
8. APPLICATION recovers her output as $De_{MS}(d_E, Y_1) = y_1$. She then delivers the garbled output wires Y_i to the jth MOBILE participant for $j \in [2, ..., N]$.
9. The jth MOBILE participant recovers his output as $En_{MS}(d_E, Y_j) = y_j$.

Figure 27: Covert Server Outsourced PFE Protocol

6.7.1 Protocol Overview

At a high level, our covert secure protocol (Figure 27) replicates the semi-honest garbling process and adds a cut-and-choose operation to ensure that the Cloud has a low probability of successfully cheating. Given a statistical security parameter λ, the Cloud and Application server execute the garbling procedure λ times. They then execute λ sets of oblivious transfers to garble the Application server's input to the computation. After both of these sets of operations are complete, the Application server selects a single execution circuit, and for the remaining check circuits, she checks the correctness of both the circuit garbling and the oblivious transfers. This ensures that the Cloud will be caught with $\epsilon = \frac{1}{\lambda}$ probability. Once this procedure is complete, each mobile device delivers his input to the Application server for the selected execution circuit. The protocol concludes as in the semi-honest protocol, with the Application server evaluating the circuit and returning the output to each mobile device.

6.7.2 Security

The security of this protocol against a covert CLOUD is essentially achieved through the addition of a cut-and-choose. While it is possible for CLOUD to improperly garble a circuit, it will be caught with a very high probability. Furthermore, because the MOBILE devices perform so few operations (sending a random seed, garbling their input, and un-garbling their output), the checks required to prevent malicious behavior are trivially added in. Finally, while there are several additional operations performed in the cut-and-choose, it can be clearly observed from the proof of the semi-honest protocol that none of these operations reveal any additional information to a semi-honest APPLICATION. We have included a complete proof of security in Section 6.8.

6.7.3 Complexity

The complexity of our covert secure protocol essentially multiplies the complexity of the semi-honest protocol by a factor of λ for both the Application Server and the Cloud. However, the complexity at the mobile parties is identical to the semi-honest model, since they

Table 12: The computational complexity of our covert outsourced PFE protocol. Note that HE signifies additively homomorphic encryptions, HA signifies homomorphic addition.

Mobile	$O(\|m_i\| + \|y_i\|)$
Application server	$\lambda[O(g) + O(g \text{ HE}) + O(g \text{ HA}) + O(k)]$
Cloud	$\lambda[O(g) + O(g \text{ HE}) + O(k)]$

still only garble input and ungarble output for a single circuit execution. These formulae are summarized in Table 12.

6.8 Covert Protocol Proof

Here we prove the security of our outsourced PFE protocol against a covert Cloud according to the following theorem.

Theorem 3. *The outsourced PFE protocol securely and privately computes a circuit C with a covert security parameter of $\epsilon = \frac{1}{\lambda}$ when the Application server is semi-honest and non-colluding, the Cloud is covert and non-colluding, and the mobile devices are malicious.*

6.8.1 Malicious Mobile

Here we construct a simulator S_M that can simulate the view and output of a malicious Mobile adversary M^*. Without loss of generality, this party represents any number of colluding Mobile devices executing the protocol. We construct this simulator through a set of hybrid experiments.

$Hyb1^{(M)}(k, x, f; r)$: This experiment is identical to $REAL^{(M)}(k, x, f; r)$ except that the experiment terminates if any of the parties controlled by M^* send an inconsistent seed s to Cloud.

Lemma 57. $REAL^{(M)}(k, x, f; r) \overset{c}{\approx} Hyb1^{(M)}(k, x, f; r)$

Proof. In the real world protocol, Cloud will terminate upon receiving inconsistent seeds from any Mobile participant, thus, these two experiments are identical. □

$Hyb2^{(M)}(k, x, f; r)$: This experiment is identical to $Hyb1^{(M)}(k, x, f; r)$ except that the experiment uses s to generate the input wire labels and recover M^*s actual input m^* to the

computation. If any of the inputs are malformed, the experiment terminates. Otherwise, the experiment delivers m^* to the trusted third party and receives $y^* = f(m^*)$ as a result.

Lemma 58. $Hyb1^{(M)}(k, x, f; r) \overset{c}{\approx} Hyb2^{(M)}(k, x, f; r)$

Proof. In the real world protocol, if any party delivers a malformed garbled input, the circuit evaluation will fail and the protocol will terminate except with a negligible probability. Otherwise, the garbled input delivered here will be the input value used in computing the garbled circuit, so the output will match the ideal world output y^*. □

$Hyb3^{(M)}(k, x, f; r)$: This experiment is identical to $Hyb2^{(M)}(k, x, f; r)$ except that the experiment uses s to generate the output wire labels and garbles y^* before returning the result to M^*.

Lemma 59. $Hyb2^{(M)}(k, x, f; r) \overset{c}{\approx} Hyb3^{(M)}(k, x, f; r)$

Proof. This hybrid holds since the garbled wire values in the real world protocol are generated deterministically based on s. In addition, since the arbitrary malicious input m^* was recovered and used in the ideal world computation, the output y^* will match in both the real and ideal execution. □

Lemma 60. $Hyb3^{(M)}(k, x, f; r)$ *runs in polynomial time.*

Proof. This lemma follows trivially since a real world execution of the protocol runs in polynomial time and each intermediate hybrid adds operations that are polynomial with respect to input and output lengths. □

We conclude the proof by letting S_M execute $Hyb3^{(M)}(k, x, f; r)$, following the real world protocol where the hybrid experiment does not differ. S_M runs M^* and controls Cloud and Application. S_M simulates the real world protocol and outputs whatever M^* outputs at the end of the simulation. From Lemma 57-60, S_M proves Theorem 3 when the Mobile parties are malicious.

6.8.2 Covert Cloud

Here we construct a simulator S_C that can simulate the view and output of a covert Cloud adversary C^*. We construct this simulator through a set of hybrid experiments.

$Hyb1^{(C)}(k, x, f; r)$: This experiment is identical to $REAL^{(C)}(k, x, f; r)$ except that rather than garble the private function $f(\cdot)$ during the $Garble_{CTH}()$ functions, the experiment garbles a random function $f'(\cdot)$ such that $\Phi_{size}(f) = \Phi_{size}(f')$.

Lemma 61. $REAL^{(C)}(k, x, f; r) \overset{c}{\approx} Hyb1^{(C)}(k, x, f; r)$

Proof. This lemma follows from the proof of Mohassel and Sadeghian [126], which states that the view of the adversary C^* can be simulated indistinguishably by a simulator S_{CTH}. Even though the garbling party is covert, the output of the oblivious extended permutation (OEP) is always a set of uniformly random strings based on the blinding values used by the experiment to hide the circuit topology. Thus, C^* cannot distinguish between garbling $f(\cdot)$ and garbling $f'(\cdot)$. □

$Hyb2^{(C)}(k, x, f; r)$: This experiment is identical to $Hyb1^{(C)}(k, x, f; r)$ except that the experiment invokes a simulator S_{OT} with a random input x' to simulate C^*s view of the oblivious transfer.

Lemma 62. $Hyb1^{(C)}(k, x, f; r) \overset{c}{\approx} Hyb2^{(C)}(k, x, f; r)$

Proof. This lemma holds based on the covert security of the OT protocol used. This guarantees that a simulator S_{OT} exists and can indistinguishably simulate C^*s view for any input value. □

$Hyb3^{(C)}(k, x, f; r)$: This experiment is identical to $Hyb2^{(C)}(k, x, f; r)$ except that the experiment uses s, which it generated according to the protocol, to check all of the garbled circuits sent by C^* according to the checking procedure in the real protocol. The experiment does one of three things:

1. If all circuits, output tables, and the result of the OT are consistent and correct, the experiment continues.

2. If exactly one circuit is incorrectly garbled (or the output table is malformed, or the output of the OT for one circuit is incorrectly generated), the experiment sends `cheat` to the trusted third party. If the trusted party returns `caught`, the experiment terminates. Otherwise, it continues.

3. If more than one circuit is incorrectly garbled (or the output tables are malformed, or the output of the OT for more than one circuit is incorrectly generated), the experiment terminates.

Lemma 63. $Hyb2^{(C)}(k, x, f; r) \stackrel{c}{\approx} Hyb3^{(C)}(k, x, f; r)$

Proof. This hybrid essentially matches the probability of failure during the cut-and-choose for each scenario:

1. If C^* runs the protocol correctly, the cut-and-choose will pass and the protocol will continue.

2. If C^* corrupts only one circuit (or output table or OT result), there is a $\frac{1}{\lambda}$ probability that it will not be caught in the cut-and-choose. This matches the probability that the trusted third party will not return `caught` if we assume $\epsilon = \frac{1}{\lambda}$.

3. If C^* corrupts multiple circuits, it will always be caught during the cut-and-choose, and so the protocol will terminate in both the real and ideal execution.

□

Lemma 64. $Hyb3^{(C)}(k, x, f; r)$ *runs in polynomial time.*

Proof. This lemma follows since a real world execution of the protocol runs in polynomial time; the simulator S_{OT} runs in polynomial time and is invoked a fixed number of times with respect to input length; the simulator S_{CTH} runs in polynomial time; and checking an additional circuit incurs a polynomial addition of operations with respect to circuit size. □

We conclude the proof by letting S_C execute $Hyb3^{(C)}(k, x, f; r)$, following the real world protocol where the hybrid experiment does not differ. S_C runs C^* and controls MOBILE

and APPLICATION. S_C simulates the real world protocol and outputs whatever C^* outputs at the end of the simulation. From Lemma 61-64, S_C proves Theorem 3 when the CLOUD is covert.

6.8.3 Semi-honest Application server

The only difference between this protocol and our semi-honest protocol is the addition of the cut-and-choose, which essentially repeats the semi-honest garbling protocol multiple times. Thus, simulating the Application server's view in this protocol reduces easily to the proof in Section 6.6.

6.9 Partially-Circuit Private Garbling

While the previous protocols provide relatively efficient and secure mechanisms for outsourcing PFE, they do not handle an arbitrarily malicious Cloud adversary, which is a critical requirement for highly sensitive applications. Unfortunately, the techniques used in current garbled circuit SMC protocols to achieve security against a malicious generator do not trivially translate to the PFE setting. In particular, many of these protocols assume that the function being evaluated will contain auxiliary circuits to perform checks of input consistency and encrypt the output of the circuit to maintain privacy. However, since the PFE setting allows the function holder to evaluate an arbitrarily chosen function, we cannot make these structural assumptions. As a result, these techniques can only be directly applied to PFE protocols where the function holder is always semi-honest.

To allow for development of protocols with stronger security guarantees against the function holder, we designed a partially-circuit private (PCP) garbling technique that allows the private function to receive input from some public preprocessing circuit, and output to another publicly available post-processing circuit. This allows us to incorporate auxiliary circuit based consistency checks into a PFE scheme while not making any assumptions about the private function being evaluated.

6.9.1 Protocol Overview

Our PCP garbling technique (Figure 28) is achieved by requiring the generator to alternate between different garbling techniques for each section of the circuit, then stitching these sections together by matching the appropriate garbled wire labels from the output of one circuit to the input of the next. Essentially, since the number of input and output wires of the private function are publicly known, they can be correlated to the output wires of a preprocessing function and the input wires to a post processing function. By simply garbling these circuits with the same wire labels, a circuit with a private central function can be generated by combining the MS13 private garbling technique with any Yao-based garbling technique as long as it has the projective property.

To evaluate the function, the evaluating party simply evaluates the preprocessing function using a typical garbled circuit evaluation protocol. Once she gets the output wire labels from this function, she can apply her blinding values used to garble the private function and continue evaluation according to MS13. Finally, since the output labels of the private circuit are not blinded, they can be directly input into the post-processing circuit, which is evaluated and output as in a typical garbled circuit protocol.

6.9.2 Security

To understand the security implications of combining two different garbled circuit primitives, we consider the fact that the wire labels for the private function are generated independently from each other as well as the wire labels used in the pre- and post-processing circuits. Thus, having knowledge of the input and output wires of the private section of the circuit, which are shared with one of the public circuits, reveals nothing to Cloud about the topology of the private portion of the computation. Furthermore, during the evaluation of the circuit, we are guaranteed from both garbling schemes that privacy is maintained as long as Application only observes one wire label for each wire in the circuit. This holds for the public circuits, which may use correlated wire labels to achieve free-XOR and other optimizations, as well as the private circuit, which simply requires the addition of the correct blinding pad to properly evaluate the circuit. We prove security of this construction

Common inputs: a computational security parameter k; circuit representations C_1 and C_2 of the preprocessing function $f_1(x_1, ..., x_n)$ and post processing function $f_2(x_1, ..., x_n)$; and the side information function $\Phi_{size}(C) = (n, m, q)$ for APPLICATION's private function C.
Private inputs: For every $i \in [2, ..., N]$, MOBILE$_i$ inputs a string m^i. APPLICATION inputs a circuit C_p that evaluates her private function $f_p(x_1, ..., x_n)$.
Outputs: APPLICATION receives a vector of blinding values B and CLOUD receives the garbled representation (G, e, d) of the composite function $f_2(f_p(f_1))$.
Circuit Garbling

1. Given random seeds s_1 and s_2, CLOUD garbled the pre and post processing circuits $Gb(1^k, C_1, s_1) \rightarrow (G_1, e_1, d_1)$ and $Gb(1^k, C_2, s_2) \rightarrow (G_2, e_2, d_2)$. For each of these circuits, there is an associated vector of input wire labels IW_i and a vector of output wire labels OW_i.

2. Using the random seed r_p, CLOUD and APPLICATION run the modified $Gb_{MS}(1^k, C_p, r_p, OW_1, IW_2) \rightarrow (G_p, B_p, e_p, d_p)$. Rather than generating new wire labels for the input and output wires of G_p, the modified garbling algorithm associates OW_1 with the input wires of G_p and IW_2 with the output wires of G_p. This sets the composite functions $En_{MS}(e_p, De(d_1, Y_1)) = Y_1$ and $En(e_2, De_{MS}(d_p, Y_p)) = Y_p$ to both be the *identity function*.

3. CLOUD receives the composite garbled circuit (G_c, e_1, d_2) and APPLICATION receives a vector of blinding values B_p for the private section of the function.

Circuit Evaluation

1. Given the garbled input values $[En(e_1, x_i)]$ for $i \in [1, ..., N]$ and G_c, APPLICATION evaluates $Ev(G_1, [En(e_1, x_i)])$ using the chosen garbled circuit evaluation technique.

2. Given the garbled wire values for $Ev(G_1, [En(e_1, x_i)])$, APPLICATION evaluates $Ev_{MS}(G_p, Ev(G_1, [En(e_1, x_i)]))$ using the MS13 evaluation algorithm.

3. Given the garbled wire values for $Ev_{MS}(G_p, Ev(G_1, [En(e_1, x_i)]))$, APPLICATION concludes by evaluating $Ev(G_2, Ev_{MS}(G_p, Ev(G_1, [En(e_1, x_i)])))$ again using the chosen evaluation algorithm.

Figure 28: Outsourced PCP garbling and evaluation protocols

applied to our malicious secure PFE outsourcing protocol in Section 6.11, but these security properties can be shown to extend to more general applications as well.

6.9.3 Efficiency Concerns

This protocol for combining private and public circuit garbling adds no overhead to any of the garbling techniques used. Since it relies only on correctly matching the output wire labels from the previous circuit to the input wires of the next circuit, each sub-circuit can be

garbled for the cost of garbling that circuit outside of the combined protocol. In addition, while the private circuit garbling requires that the wire labels be chosen independently at random and that all gates be garbled as NAND gates, this same restriction does not apply to the public portions of the circuit. As our security discussion shows, any of the recently developed optimizations for Yao circuit garbling can be applied to the preprocessing and post-processing circuits (e.g., garbled row reduction, free-XOR, etc.)

6.9.4 Additional Applications

While we apply PCP garbling for the specific purpose of protocol security enhancement, our technique promises applicability for an array of practical computation needs. For a variety of data mining scenarios, data must be aggregated to preserve privacy before it is processed by research or advertising agencies. By making the necessary preprocessing a public circuit, data owners can be assured that their data is properly anonymized before any proprietary or sensitive analytic functions are applied using the private circuit. To summarize, our PCP garbling technique is applicable in any PFE application where multiple parties need assurance that the data is prepared or finalized using specific operations.

6.10 Malicious Server Outsourced PFE

To apply the partially private circuit garbling to our previous PFE outsourcing techniques, we can now apply the k-probe-resistant input encoding technique and the 2-Universal output hash as implemented by shelat and Shen [151] in our preprocessing function. This ensures that all parties participating in the computation will know and can verify that the necessary checks are being performed within the garbled circuit. We present our protocol and prove security in the presence of a semi-honest Application server that is in possession of the private function. However, the addition of an aut-secure (defined by Bellare et al. [16]) garbling scheme and an efficient zero-knowledge circuit commitment would allow this protocol to be secure even in the presence of a malicious function holder. This is because whatever arbitrary function is chosen by the Application server to be evaluated, to prove security the function must be recoverable by the simulator and sent to the trusted third party to be evaluated in the ideal world. If this function is committed and verified

to have been executed by the Application server, then the output distributions between the real world and ideal world will be computationally indistinguishable. Furthermore, the function holder must not be able to tamper with the output of the garbled circuit, which is prevented by the aut-secure garbling scheme. While this adversary model appears flawed in practice when the function holder both defines the function *and* receives the output of that function, it provides a useful guarantee when *only* the mobile parties receive output from the computation.

6.10.1 Protocol Overview

The malicious secure outsourced PFE protocol (Figure 29) begins by having all mobile participants commit to their inputs for a set of λ garbled preprocessing circuits, where λ is a statistical security parameter. Once these values are committed, the generating party will garble λ copies of the preprocessing function, post processing function, and private function, using the appropriate garbling protocol for each section. Once the Application server possesses the garbled circuits and has her input delivered for each via oblivious transfer, she selects a fraction of the circuits to be opened using cut-and-choose. Once these circuits are verified, the mobile parties decommit their inputs corresponding to the remaining evaluation circuits, and the Application server evaluates the remaining unopened garbled circuits. Finally, the Application server concludes by taking the majority output value from all of the evaluation circuits and delivering this value to the mobile devices to conclude the computation.

6.10.2 Security

The primary requirement for achieving malicious security over covert security is in modifying the cut-and-choose to include *multiple* evaluation circuits. This modification allows us to reduce the probability of cheating to a computationally negligible probability. Our cut-and-choose technique is derived from the two-party garbled circuit protocol developed by shelat and Shen [150], which gives $2^{-0.32\lambda}$ probability of an adversary successfully cheating. Given our computational security parameter k, we use this bound to set $\lambda = \frac{k}{0.32}$.

Common inputs: a computational security parameter k, a statistical security parameter λ, a commitment scheme $com_c(x)$ with key c and message x, and the side information function $\Phi_{size}(C) = (n, m, q)$ for Application's private function C. Here, n is the length of a single party's input, m is the length of a single party's output, and q is the number of gates in C. All Mobile participants share a common random seed s where $|s| = k$.
Private inputs: For every $i \in [2, ..., N]$, Mobile$_i$ inputs a string m^i. Application inputs a bit string a and a circuit C that evaluates her private function $f(a, m^2, ..., m^N)$.
Outputs: Application receives the output y_1 and Mobile$_i$ receives the output y_i for $i \in [2, ..., N]$.

1. The Mobile devices deliver s to Cloud.

2. Cloud ensures that every copy of s received is identical. If any of the values differs, Cloud aborts. Otherwise, Cloud uses a pseudorandom generator seeded with s to produce random seeds r_i for $i \in [1, ..., \lambda]$.

3. Every Mobile participant generates the same set of seeds r_i for $i \in [1, ..., \lambda]$. The j^{th} Mobile then generate a one-time pad $b_j \in \{0,1\}^m$ for $j \in [2, ..., N]$ to blind his output from the evaluated circuits. Each then garbles his input using each seed, so that the j^{th} mobile party produces $En(e_i, m^j||b_j)$ for $i \in [1, ..., \lambda]$. Finally, Mobile generates commitment keys c_i for $i \in [1, ..., \lambda]$ and commits to each garbled input as $com_{c_i}(En(e_i, m^j||b_j))$ for $i \in [1, ..., \lambda]$. Mobile delivers all of his commitments to Application.

4. Cloud and Application agree on the preprocessing and post-processing functions. These parties jointly establish a 2-Universal hash matrix $\mathbf{H} \in \{0,1\}^{k \times n}$, and Application generates her k-probe-resistant matrix $\mathbf{M}$. She then defines an input vector $\overline{a}$ such that $\mathbf{M} \cdot \overline{a} = a$. They then define the preprocessing function $f_1(\overline{a}, m^2, ..., m^N) = \{\mathbf{M} \cdot \overline{a}, m^2, ..., m^N\}$ and the output function to be $f_2(f(a, m^2, ..., m^N)) = \{y_1, [y_2 \oplus b_2, \mathbf{H} \cdot m^2], ..., [y_N \oplus b_N, \mathbf{H} \cdot m^N]\}$. They set C_1 and C_2 to be the circuit representations of these functions.

5. For every index $[i \in 1, ..., \lambda]$, Cloud and Application run $Gb_{PCP}(1^k, C_1, C_2, C, r_i) \rightarrow (G_i, B_i, e_i, d_i)$.

Figure 29: Malicious Server Outsourced PFE Protocol – Part 1

6. For every $i \in [1, ..., \lambda]$, Application and Cloud run k instances of a malicious-secure oblivious transfer protocol (using OT-extensions). After the protocol, Application receives her garbled input $En(e_i, \overline{a})$.

7. Application selects a random set of indices $[E] \in [1, ..., \lambda]$ where $|[E]| = \frac{2\lambda}{5}$. For the indices $i \in [1, ..., \lambda] \setminus [E]$, Cloud reveals r_i. Application runs $Gb_{PCP}(1^k, C_1, C_2, C, r_i) \rightarrow (G'_i, B'_i, e'_i, d'_i)$ and checks:

   (a) $G_i = G'_i$

   (b) $d_i = d'_i$

   (c) $En(e_i, \overline{a}) = En(e'_i, \overline{a})$

   If any of these checks fails for any index i, Application aborts.

8. Every Mobile participant decommits its garbled input by delivering c_i for $i \in [E]$.

9. Application For every $i \in [E]$, Application runs $Ev_{PCP}(G_i, B_i, En(e_i, \overline{a}), En(e_i, m^2||b^2), ..., En(e_i, m^N||b^N)) = Y_1, Y_2, ..., Y_N$

10. For all $i \in [E]$, Application recovers $De(d_i, go_1) = y_1$ and $De(d_i, go_j) = y_j \oplus b_j, \mathbf{H} \cdot m^j$ for all $j \in [2, ..., N]$. For each $j \in [2, ..., N]$, Application ensures that the hash of the input $\mathbf{H} \cdot m^j$ is the same across every evaluated circuit in $[E]$ and aborts otherwise.

11. Application takes a majority vote from all evaluation circuits $i \in [E]$ to determine the output $y_1, y_2 \oplus b_2, ..., y_N \oplus b_N$ for all parties. If no majority exists, Application aborts. Otherwise, she delivers $y_j \oplus b_j$ to the j^{th} Mobile party.

12. The j^{th} Mobile party recovers his output $y_j = (y_j \oplus b_j) \oplus b_j$.

Figure 30: Malicious Server Outsourced PFE Protocol – Part 2

This cut-and-choose technique requires additional verification to ensure that the Mobile participants provide consistent inputs to all evaluation circuits, as well as taking the majority output from the evaluation circuits to prevent the Cloud from learning anything about a party's input based on differing circuit outputs from the evaluation circuits. By applying our partial PFE garbling technique, we can incorporate existing techniques from traditional malicious secure garbled circuit protocols to combat these attacks. In particular, we ensure input consistency using shelat and Shen's 2-Universal hash construction [151], output privacy during the majority vote with additional one-time pads, and selective failure prevention using the k-probe-resistant encoding as implemented by shelat and Shen [151]. However, our protocol could easily be modified to incorporate new consistency checks as they are developed. We formally prove the security of our construction in Section 6.11.

Recent work in two-party SMC has produced protocols with more efficient cut-and-choose techniques, such as Lindell's auxiliary circuit technique [109]. However, applying this techniques to the outsourced PFE setting remains an open challenge. In particular, Lindell's technique of punishing the generator by exposing his outputs to the evaluator does not translate into a setting where the generator is responsible for other parties' inputs. In our setting, Cloud could potentially cheat, which would result in an honest Mobile party's input being revealed to Application.

6.10.3 Complexity

This protocol magnifies the complexity of our covert secure protocol with respect to three parameters: the size of the circuit representations of f_1 and f_2, as well as the increase in λ to ensure security against a malicious Cloud ($\lambda = 16$ is common in practice for covert security, with $\lambda = 256$ for malicious security under our cut-and-choose parameters.) In this protocol, Cloud is tasked with garbling λ copies of the private circuit C and both public functions C_1 and C_2. Application is then responsible for checking $\frac{3\lambda}{5}$ complete circuits (preprocessing, private, and postprocessing), as well as evaluating $\frac{2\lambda}{5}$ remaining circuits, according to the optimal cut-and-choose parameters used by shelat and Shen [150]. For a detailed representation of these operations, see Table 13.

Table 13: The computational complexity of our malicious outsourced PFE protocol. Note that HE signifies additively homomorphic encryptions, HA signifies homomorphic addition.

Mobile	$O(2\|m_i\| + \|y_i\|)$
Application server	$\lambda[O(\|C_1\| + g + \|C_2\|) + O(g \text{ HE}) + O(g \text{ HA}) + O(k)]$
Cloud	$\lambda[O(\|C_1\| + g + \|C_2\|) + O(g \text{ HE}) + O(k)]$

6.10.4 Black Box PFE Outsourcing

Jakobsen et al. [89], as well as our work in Chapter 5, show that the concept of auxiliary circuits that we apply here can be extended for efficiently outsourcing SMC protocols in a completely black-box manner. However, additional research into varied adversary models and computation techniques is necessary before black-box outsourcing can be applied to PFE protocols. Our partial PFE garbling technique offers a useful first step in developing a completely black-box approach to outsourcing PFE. However, it can only be applied to garbled circuit-based PFE schemes. To achieve security against malicious adversaries using this generic technique, a two-party PFE protocol using garbled circuits must first be developed. The only existing PFE protocol that has been demonstrated secure against malicious adversaries is the secret-sharing based scheme of Mohassel et al. [125]. Our outsourcing protocol can be reduced to a two-party PFE protocol, but is only secure against a malicious circuit generator, while the function holder must be semi-honest.

6.11 Malicious Protocol Proof

Here we prove the security of our outsourced PFE protocol against a malicious Cloud according to the following theorem.

Theorem 4. *The outsourced PFE protocol securely and privately computes a circuit C when the Application server is semi-honest and non-colluding, the Cloud is malicious and non-colluding, and the mobile devices are malicious.*

6.11.1 Malicious Mobile device

Here we construct a simulator S_M that can simulate the view and output of a malicious Mobile adversary M^*. Without loss of generality, this party represents any number of colluding Mobile devices executing the protocol. We construct this simulator through a

set of hybrid experiments.

$Hyb1^{(M)}(k, x, f; r)$: This experiment is identical to $REAL^{(M)}(k, x, f; r)$ except the experiment terminates if any of the parties controlled by M^* send an inconsistent seed s.

Lemma 65. $REAL^{(M)}(k, x, f; r) \stackrel{c}{\approx} Hyb1^{(M)}(k, x, f; r)$

Proof. In the real world protocol, CLOUD will terminate upon receiving inconsistent seeds from any MOBILE participant, thus, these two experiments are identical. □

$Hyb2^{(M)}(k, x, f; r)$: This experiment is identical to $Hyb1^{(M)}(k, x, f; r)$ except that after M^* decommits his inputs, the experiment uses s to generate the input wire labels and recover M^*s actual input m^* to the computation. If any of the decommitted inputs are inconsistent or malformed, the experiment terminates. Otherwise, the experiment delivers m^* to the trusted third party and receives $y^* = f(m^*)$ as a result.

Lemma 66. $Hyb1^{(M)}(k, x, f; r) \stackrel{c}{\approx} Hyb2^{(M)}(k, x, f; r)$

Proof. In the real world, if any party delivers inconsistent input, the 2-Universal hash will reveal the inconsistency with all but negligible probability. In addition, malformed garbled input will cause the evaluation to fail and the protocol will again terminate except with a negligible probability. Otherwise, the garbled input delivered here will be the input value used in the garbled circuit, so the output will match the ideal world output y^*. □

$Hyb3^{(M)}(k, x, f; r)$: This experiment is identical to $Hyb2^{(M)}(k, x, f; r)$ except that the experiment uses s to generate the output wire labels and returns the garbled y^* to M^*.

Lemma 67. $Hyb2^{(M)}(k, x, f; r) \stackrel{c}{\approx} Hyb3^{(M)}(k, x, f; r)$

Proof. This hybrid holds since the garbled wire values in the real world protocol are generated deterministically based on s. In addition, since the arbitrary malicious input m^* was recovered and used in the ideal world computation, the output y^* will match in both the real and ideal execution. □

Lemma 68. $Hyb3^{(M)}(k, x, f; r)$ *runs in polynomial time.*

Proof. This lemma follows trivially since a real world execution of the protocol runs in polynomial time and each intermediate hybrid adds operations that are polynomial with respect to input and output lengths. □

We conclude the proof by letting S_M execute $Hyb3^{(M)}(k, x, f; r)$, following the real world protocol where the hybrid experiment does not differ. S_M runs M^* and controls CLOUD and APPLICATION. S_M simulates the real world protocol and outputs whatever M^* outputs at the end of the simulation. From Lemma 65-68, S_M proves Theorem 4 when the MOBILE parties are malicious.

6.11.2 Malicious Cloud

Here we construct a simulator S_C that can simulate the view and output of a malicious CLOUD adversary C^*. We construct this simulator through a set of hybrid experiments.

$Hyb1^{(C)}(k, x, f; r)$: This experiment is identical to $REAL^{(C)}(k, x, f; r)$ except that rather than garble the private function $f(\cdot)$ during the $Gb_{MS}()$ functions, the experiment garbles a random function $f'(\cdot)$ where $\Phi_{size}(f') = \Phi_{size}(f)$.

Lemma 69. $REAL^{(C)}(k, x, f; r) \overset{c}{\approx} Hyb1^{(C)}(k, x, f; r)$

Proof. This lemma follows from the proof of Mohassel and Sadeghian [126], which constructs a simulator S_{CTH} that can simulate C^*s view of the real execution given only $\Phi_{size}(f)$. The output of S_{CTH} after the oblivious extended permutation (OEP) is a set of uniformly random strings based on the blinding values used by the experiment to hide the circuit topology. Thus, C^* cannot distinguish between garbling $f(\cdot)$ and garbling $f'(\cdot)$. □

$Hyb2^{(C)}(k, x, f; r)$: This experiment is identical to $Hyb1^{(C)}(k, x, f; r)$ except that the experiment invokes a simulator S_{OT} to simulate C^*s view of the oblivious transfer.

Lemma 70. $Hyb1^{(C)}(k, x, f; r) \overset{c}{\approx} Hyb2^{(C)}(k, x, f; r)$

Proof. This lemma holds based on the malicious security of the OT protocol used. This guarantees that a simulator S_{OT} exists, can indistinguishably simulate C^*s view of a real execution, and can recover C^*s input to the OT w_i^0, w_i^1 for $i \in [1, ..., n]$. □

$Hyb3^{(C)}(k, x, f; r)$: This experiment is identical to $Hyb2^{(C)}(k, x, f; r)$ except that the experiment uses s, which it generated according to the protocol, to check the garbled circuits sent by C^*. The experiment does one of three things:

1. If any of the circuits G_j, output tables d_j, and the result of the OT $w^0_{i,j}, w^1_{i,j}$ for $i \in [1, ..., n]$ and for $j \in [1, ..., \lambda] \setminus [E]$ are not consistent and correct, the experiment terminates.

2. If a majority of the circuits G_j, output tables d_j, and the result of the OT $w^0_{i,j}, w^1_{i,j}$ for $i \in [1, ..., n]$ and for $j \in [E]$ are not consistent and correct, the experiment terminates.

3. Otherwise, the experiment continues.

Lemma 71. $Hyb2^{(C)}(k, x, f; r) \overset{c}{\approx} Hyb3^{(C)}(k, x, f; r)$

Proof. This hybrid essentially matches the probability of failure during the cut-and-choose for each scenario:

1. If C^* corrupts a circuit in the check circuits $[1, ..., \lambda] \setminus [E]$, the experiment terminates.

2. If C^* corrupts a majority of the evaluation circuits in $[E]$, the experiment terminates. Based on the proof by shelat and Shen [150], this happens with negligible probability.

3. Otherwise, execution continues with a majority of evaluation circuits correctly constructed.

□

Lemma 72. $Hyb3^{(C)}(k, x, f; r)$ *runs in polynomial time.*

Proof. This lemma follows since a real world execution of the protocol runs in polynomial time; the simulator S_{OT} runs in polynomial time and is invoked a fixed number of times with respect to input length; S_{CTH} runs in polynomial time; and checking the additional circuits incurs a polynomial addition of operations with respect to circuit size. □

We conclude the proof by letting S_C execute $Hyb3^{(C)}(k, x, f; r)$, following the real world protocol where the hybrid experiment does not differ. S_C runs C^* and controls MOBILE

and APPLICATION. S_C simulates the real world protocol and outputs whatever C^* outputs at the end of the simulation. From Lemma 69-72, S_C proves Theorem 4 when the CLOUD is malicious.

6.11.3 Semi-honest Application server

Here we construct a simulator S_A that can simulate the view of a semi-honest APPLICATION adversary A^*. We construct this simulator through a set of hybrid experiments.

$Hyb1^{(A)}(k, x, f; r)$: This experiment is identical to $REAL^{(A)}(k, x, f; r)$ except that the experiment sends A^*s inputs $f(\cdot), x$ to the trusted third party.

Lemma 73. $REAL^{(A)}(k, x, f; r) \stackrel{c}{\approx} Hyb1^{(A)}(k, x, f; r)$

Proof. Since A^* is semi-honest, the experiment invokes the trusted party using the input function $f(\cdot)$ and the input data x that it provided to A^* at the start of the experiment. The value returned by the trusted third party $y = f(x)$ will be identical to the value output by the circuit evaluated in the real world. □

$Hyb2^{(A)}(k, x, f; r)$: This experiment is identical to $Hyb1^{(A)}(k, x, f; r)$ except that the experiment commits to a random input m' rather than the real MOBILE input m.

Lemma 74. $Hyb1^{(A)}(k, x, f; r) \stackrel{c}{\approx} Hyb2^{(A)}(k, x, f; r)$

Proof. Indistinguishability here is based on two properties. First, the hiding commitment scheme guarantees the commitments are indistinguishable between the ideal and real world. Second, the 2-Universal hash construction of shelat and Shen [151] guarantees the hashes that are output by the circuit for consistency checking are indistinguishable for any two inputs. □

$Hyb3^{(A)}(k, x, f; r)$: This experiment is identical to $Hyb2^{(A)}(k, x, f; r)$ except that the experiment uses the random coins given to A^* to determine $[E]$. During the $Gb_{PCP}()$ execution, for all circuits $F \in [E]$, the experiment simulates the view of A^* using simulators S_1, S_{MS}, S_2.

Lemma 75. $Hyb2^{(A)}(k, x, f; r) \stackrel{c}{\approx} Hyb3^{(A)}(k, x, f; r)$

Proof. This follows from the prv.sim security of the preprocessing and post-processing garbling schemes, as well as the prv.sim security of Gb_{MS} proven in Theorem 1, all of which guarantee a simulator S_1, S_{CTH}, S_2 that can simulate the adversary's view of a real circuit and garbled inputs given only Φ_{circ} and $y = f_2(f(f_1(x)))$. □

Lemma 76. $Hyb3^{(A)}(k, x, f; r)$ *runs in polynomial time.*

Proof. This lemma follows trivially since the experiment runs the real world protocol and a set of simulators S_1, S_2, S_{CTH} that all run in polynomial time. □

We conclude the proof by letting S_A execute $Hyb3^{(A)}(k, x, f; r)$, following the real world protocol where the hybrid experiment does not differ. S_A runs A^* and controls CLOUD and MOBILE. S_A simulates the real world protocol and outputs whatever A^* outputs at the end of the simulation. From Lemma 73-76, S_A proves Theorem 4 when the APPLICATION party is semi-honest.

6.12 Conclusion

The outsourced SMC model developed in the previous chapters has shown that SMC protocols can be efficiently applied to the mobile platform for a variety of applications. However, SMC outsourcing protocols require that the function being evaluated be public to all participants, which prohibits their use in a variety of government and commercial applications. This chapter develops the first efficient outsourced PFE protocols that allow for private applications to be evaluated securely using mobile input sources. By combining existing outsourcing techniques with a novel partially-circuit private garbling technique, we show that security can be maintained against malicious mobile devices and covert or malicious Cloud providers. These protocols allow for a range of new, real-world mobile applications that could not be securely executed using previous outsourcing protocols.

CHAPTER VII

FUTURE WORK AND CONCLUSION

7.1 Future Work

While this work has demonstrated significant progress towards practical mobile SMC, there are still a variety of open problems that promise to produce improvements in outsourcing techniques and SMC protocols in general.

7.1.1 Practically Motivated Applications

As new SMC techniques are continually being developed, a hotly debated topic in the research community is how to move these prototype techniques into production use. While many compelling privacy-preserving example applications have been proposed, many of these applications lack the financial incentive for industry to further develop them into a live application. For example, maintaining privacy of location queries and search terms from a large provider like Google is a good application for the user. However, it removes Google's ability to monetize this information in the form of advertising, negating their financial motivation to actually implement the application.

To solve this problem, future research should focus on identifying applications that provide mutual benefit for all parties involved in the computation. For example, legal restrictions on medical record privacy is making the transition to cloud-based record storage costly and slow [138]. In addition, as medical professionals begin using smartphones and tablets to access these records, SMC offers a compromise that allows medical professionals the convenience of access they require while still maintaining the legally required privacy protection. As a second example, biometric authentication presents unique potential for easy to use authentication, but presents significant problems with revocation. In this setting, added privacy protection is necessary to prevent the leakage and invalidation of biometric credentials. While academic research may continue to make progress towards

more efficient SMC constructions, we cannot address the practical issues associated with at-scale implementation until a strongly motivated practical application is put into production use.

7.1.2 Optimal Combination Protocols

After identifying potential real-world applications for SMC protocols, the research community will have the opportunity to improve SMC performance by optimizing protocols that meet the specific needs of these practically motivated applications. As recent work has shown, combining SMC protocols can lead to significantly improved overall performance depending on the underlying function being evaluated [77, 54]. However, to maximize these performance gains, these combination SMC protocols must be tailored towards specific functions. These custom optimizations for real-world applications may provide the last necessary improvements to move SMC from an experimentally feasible construction to a practically deployed industrial tool.

7.2 Conclusion

As mobile computing continues to become more commonplace, privacy-preserving computation techniques are necessary to allow users to take advantage of mobile applications without the threat of their private information being exposed. While SMC protocols provide us with tools for such privacy-preserving computation on the desktop platform, these protocols too resource-intensive for the computational, power, and bandwidth limitations inherent to the mobile platform. In this dissertation, we have developed a wide range of techniques for outsourcing the most costly operations of SMC protocols to an untrusted Cloud provider, and have shown that these techniques can be applied in practice to many real-world applications. Our first two protocols demonstrate that outsourcing garbled circuit computation can achieve significant increases in the efficiency of SMC executed from a mobile device, and provided foundational techniques for future developments in outsourcing protocols. Our black box construction provides a forward compatible technique for turning any two-party SMC protocol into an outsourced protocol. This generic construction will continue to be applicable as new SMC protocols are developed in the future. Finally,

our outsourced PFE protocols establish a set of protocols that provide a range of security guarantees, allowing for application-specific tradeoffs between security and efficiency. Furthermore, these protocols provide the strongest security against the third party Cloud provider, preventing it from learning anything about the computation beyond the circuit size. As a whole, this body of work has moved the state of the art in mobile SMC from completely infeasible to near-practical, and promises to allow for deployable SMC protocols as new techniques continue to be developed.

REFERENCES

[1] Abadi, M. and Feigenbaum, J., "Secure circuit evaluation," *Journal of Cryptology*, vol. 2, no. 1, pp. 1–12, 1990.

[2] Alperin-Sheriff, J. and Peikert, C., "Faster bootstrapping with polynomial error," in *CRYPTO*, 2014.

[3] Anderson, D. P., "BOINC: A system for public resource computing and storage," in *Proceedings of the IEEE/ACM International Workshop on Grid Computing*, 2004.

[4] Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., and Werthimer, D., "SETI@home: an experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[5] Armbrust, M., Stoica, I., Zaharia, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M., "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[6] Asharov, G., Lindell, Y., Schneider, T., and Zohner, M., "More efficient oblivious transfer and extensions for faster secure computation," in *Proceedings of the ACM conference on Computer and Communications Security*, 2013.

[7] Atallah, M. J., Bykova, M., Li, J., Frikken, K. B., and Topkara, M., "Private collaborative forecasting and benchmarking," in *Workshop on Privacy in the Electronic Society*, 2004.

[8] Atallah, M. J. and Frikken, K. B., "Securely outsourcing linear algebra computations," in *Proceedings of the ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2010.

[9] Aumann, Y. and Lindell, Y., "Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries," *Journal of Cryptology*, vol. 18, no. 3, pp. 554–343, 2010.

[10] Barni, M., Failla, P., Kolesnikov, V., Lazzeretti, R., Sadeghi, A.-R., and Schneider, T., "Secure evaluation of private linear branching programs with medical applications," in *ESORICS*, 2009.

[11] Beaver, D., "Precomputing oblivious transfer," in *CRYPTO*, 1995.

[12] Beaver, D., "Efficient multiparty protocols using circuit randomization," in *CRYPTO*, 1991.

[13] Beaver, D., "Correlated pseudorandomness and the complexity of private computations," in *Proceedings of the Annual ACM Symposium on Theory of Computing*, 1996.

[14] Beaver, D., "Server-assisted cryptography," in *Proceedings of the workshop on New security paradigms (NSPW)*, 1998.

[15] Bellare, M., Hoang, V. T., Keelveedhi, S., and Rogaway, P., "Efficient garbling from a fixed-key blockcipher," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2013.

[16] Bellare, M., Hoang, V. T., and Rogaway, P., "Foundations of garbled circuits," in *Proceedings of the ACM conference on Computer and communications security (CCS)*, 2012.

[17] Bellare, M. and Micali, S., "Non-interactive oblivious transfer and applications," in *CRYPTO*, 1990.

[18] Ben-Or, M., Goldwasser, S., and Wigderson, A., "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the annual ACM symposium on Theory of computing*, 1988.

[19] Bendlin, R., Damgård, I., Orlandi, C., and Zakarias, S., "Semi-homomorphic encryption and multiparty computation," in *Proceedings of the Annual international conference on Theory and applications of cryptographic techniques*, 2011.

[20] Blanton, M., Steele, A., and Alisagari, M., "Data-oblivious graph algorithms for secure computation and outsourcing," in *Proceedings of the ACM SIGSAC Symposium on Information, Computer and Communications Security*, 2013.

[21] Bogdanov, D., Laur, S., and Willemson, J., "Sharemind: A Framework for Fast Privacy-Preserving Computations," in *Proceedings of the European Symposium on Research in Computer Security*, 2008.

[22] Boneh, D., Gentry, C., Gorbunov, S., Halevi, S., Nikolaenko, V., Segev, G., Vaikuntanathan, V., and Vinayagamurthy, D., "Fully key-homomorphic encryption, arithmetic abe and compact garbled circuits," in *EUROCRYPT*, 2014.

[23] Brakerski, Z., "Fully homomorphic encryption without modulus switching from classical gapsvp," in *CRYPTO*, 2012.

[24] Brakerski, Z., Gentry, C., and Vaikuntanathan, V., "(leveled) fully homomorphic encryption without bootstrapping," in *Proceedings of the Innovations in Theoretical Computer Science Conference*, 2012.

[25] Brakerski, Z. and Vaikuntanathan, V., "Fully homomorphic encryption from ring-lwe and security for key dependent messages," in *CRYPTO*, 2011.

[26] Breslin, J. and Tashenberg, C. B., *Distributed processing systems : end of the mainframe era?* AMACOM, 1978.

[27] Brickell, J., Porter, D. E., Shmatikov, V., and Witchel, E., "Privacy-preserving remote diagnostics," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2007.

[28] Brickell, J. and Shmatikov, V., "Privacy-preserving graph algorithms in the semi-honest model," in *Proceedings of the international conference on Theory and Application of Cryptology and Information Security*, 2005.

[29] Bringer, J., Chabanne, H., Patey, A., Favre, M., Schneider, T., and Zohner, M., "GSHADE: Faster privacy-preserving distance computation and biometric identification," in *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security*, 2014.

[30] Buyya, R., Yeo, C. S., and Venugopal, S., "Market-oriented cloud computing: Vision, hype, and reality for delivering IT services as computing utilities," in *Proceedings of the IEEE International Conference on High Performance Computing and Communications*, 2008.

[31] Canetti, R., Lindell, Y., Ostrovsky, R., and Sahai, A., "Universally composable two-party and multi-party secure computation," in *Proceedings of the annual ACM symposium on Theory of computing*, 2002.

[32] Carter, H., Amrutkar, C., Dacosta, I., and Traynor, P., "For your phone only: custom protocols for efficient secure function evaluation on mobile devices," *Journal of Security and Communication Networks (SCN)*, vol. 7, no. 7, pp. 1165–1176, 2014.

[33] Carter, H., Lever, C., and Traynor, P., "Whitewash: Outsourcing Garbled Circuit Generation for Mobile Devices," in *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2014.

[34] Carter, H., Mood, B., Traynor, P., and Butler, K., "Secure Outsourced Garbled Circuit Evaluation for Mobile Devices," in *Proceedings of the USENIX Security Symposium*, 2013.

[35] Carter, H., Mood, B., Traynor, P., and Butler, K., "Outsourcing secure two-party computation as a black box (short paper)," in *Proceedings of the International Conference on Cryptology and Network Security (CANS)*, 2015.

[36] Catrina, O. and Hoogh, S., "Improved primitives for secure multiparty integer computation," in *Security and Cryptography for Networks*, 2010.

[37] Catrina, O. and Saxena, A., "Secure computation with fixed-point numbers," in *Financial Cryptography and Data Security*, 2010.

[38] Champine, G. A., Coop, R. D., and Heinselman, R., *Distributed computer systems: impact on management, design, and analysis*. North-Holland Pub. Co., 1980.

[39] Chaum, D., Crépeau, C., and Damgard, I., "Multiparty unconditionally secure protocols," in *Proceedings of the annual ACM symposium on Theory of computing*, 1988.

[40] Choi, S. G., Hwang, K.-W., Katz, J., Malkin, T., and Rubenstein, D., "Secure Multi-Party Computation of Boolean Circuits with Applications to Privacy in On-Line Marketplaces," in *RSA Cryptographers' Track*, 2012.

[41] Choi, S. G., Katz, J., Kumaresan, R., and Zhou, H.-S., "On the security of the "free-XOR" technique," in *Proceedings of the international conference on Theory of Cryptography*, 2012.

[42] Chor, B., Goldreich, O., Kushilevitz, E., and Sudan, M., "Private information retrieval," in *Proceedings of the Annual Symposium on Foundations of Computer Science*, 1995.

[43] Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., and Patti, A., "CloneCloud: elastic execution between mobile device and cloud," *Proceedings of EuroSys*, 2011.

[44] comScore, "comScore Reports February 2013 U.S. Smartphone Subscriber Market Share." `http://www.comscore.com/Insights/Press_Releases/2013/4/comScore_Reports_February_2013_U.S._Smartphone_Subscriber_Market_Share`, 2013.

[45] Cramer, R., Damgård, I., and Nielsen, J. B., "Multiparty computation from threshold homomorphic encryption," in *EUROCRYPT*, 2001.

[46] Damgård, I., Keller, M., and Larraia, E., "Implementing AES via an actively/covertly secure dishonest-majority MPC protocol," in *Proceedings of the International Conference on Security and Cryptography for Networks*, 2012.

[47] Damgård, I., Pastro, V., Smart, N., and Zakarias, S., "Multiparty computation from somewhat homomorphic encryption," in *CRYPTO*, 2012.

[48] Damgård, I., Geisler, M., and Nielsen, J. B., "From passive to covert security at low cost," in *Proceedings of the 7th international conference on Theory of Cryptography*, 2010.

[49] Damgård, I. and Ishai, Y., "Scalable secure multiparty computation," in *CRYPTO*, 2006.

[50] Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., and Smart, N. P., "Practical covertly secure MPC for dishonest majority, or: Breaking the SPDZ limits," in *Computer Security–ESORICS*, 2013.

[51] Damgård, I. and Nielsen, J. B., "Scalable and unconditionally secure multiparty computation," in *CRYPTO*, 2007.

[52] Damgård, I. and Orlandi, C., "Multiparty computation for dishonest majority: From passive to active security at low cost," in *CRYPTO*, 2010.

[53] Demmler, D., Schneider, T., and Zohner, M., "Ad-hoc secure two-party computation on mobile devices using hardware tokens," in *Proceedings of the USENIX Security Symposium*, 2014.

[54] Demmler, D., Schneider, T., and Zohner, M., "ABY – a framework for efficient mixed-protocol secure two-party computation," in *Proceedings of the ISOC Symposium on Network and Distributed Systems Security*, 2015.

[55] Ekanayake, J. and Fox, G., "High performance parallel computing with cloud and cloud technologies," in *Proceedings of the International Conference on Cloud Computing (CloudComp)*, 2009.

[56] Foster, I. and Kesselman, C., *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.

[57] Frederiksen, T. K., Jakobsen, T. P., Nielsen, J. B., Nordholt, P. S., and Orlandi, C., "MiniLEGO: Efficient secure two-party computation from general assumptions," in *EUROCRYPT*, 2013.

[58] Frederiksen, T. K. and Nielsen, J. B., "Fast and maliciously secure two-party computation using the GPU," in *Applied Cryptography and Network Security*, 2013.

[59] Garay, J., Wichs, D., and Zhou, H., "Somewhat non-committing encryption and efficient adaptively secure oblivious transfer," in *CRYPTO*, 2009.

[60] Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., and Waters, B., "Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits," in *IEEE Annual Symposium on Foundations of Computer Science*, 2013.

[61] Gennaro, R., Gentry, C., and Parno, B., "Non-interactive verifiable computation: Outsourcing computation to untrusted workers," in *CRYPTO*, 2010.

[62] Gentry, C., *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[63] Gentry, C., "Fully homomorphic encryption using ideal lattices," in *Proceedings of the Annual ACM Symposium on Theory of Computing*, 2009.

[64] Gentry, C., Halevi, S., and Smart, N. P., "Homomorphic evaluation of the AES circuit," in *CRYPTO*, 2012.

[65] Gilboa, N., "Two party RSA key generation," in *Crypto*, 1999.

[66] Goldreich, O., *Foundations of Cryptography: Volume 2 – Basic Applications*. Cambridge Univ. Press, 2004.

[67] Goldreich, O., Micali, S., and Wigderson, A., "How to play any mental game," in *Proceedings of the Annual ACM Symposium on Theory of Computing*, 1987.

[68] Goldreich, O. and Kahan, A., "How to construct constant-round zero-knowledge proof systems for np," *Journal of Cryptology*, vol. 9, pp. 167–189, 1995.

[69] Goldwasser, S., Kalai, Y., Popa, R., Vaikuntanathan, V., and Zeldovich, N., "Succinct functional encryption and applications: Reusable garbled circuits and beyond," in *Proceedings of the Annual ACM Symposium on Theory of Computing*, 2013.

[70] Gorbunov, S., Vaikuntanathan, V., and Wichs, D., "Leveled fully homomorphic signatures from standard lattices," in *Proceedings of the Annual ACM Symposium on Theory of Computing*, 2015.

[71] Gordon, S. D., Katz, J., Kolesnikov, V., Labs, A.-l. B., Krell, F., and Raykova, M., "Secure Two-Party Computation in Sublinear (Amortized) Time," in *Proceedings of the ACM conference on Computer and communications security (CCS)*, 2012.

[72] Goyal, V., Mohassel, P., and Smith, A., "Efficient two party and multi party computation against covert adversaries," in *EUROCRYPT*, 2008.

[73] Green, M., Hohenberger, S., and Waters, B., "Outsourcing the Decryption of ABE Ciphertexts," in *Proceedings of the USENIX Security Symposium*, 2011.

[74] Harms, R. and Yamartino, M., "The economics of the cloud," tech. rep., Microsoft, November 2010.

[75] Hazay, C. and Lindell, Y., "Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries," *Journal of Cryptology*, vol. 23, no. 3, pp. 422–456, 2008.

[76] Hazay, C. and Lindell, Y., *Efficient Secure Two-Party Protocols: Techniques and Constructions*. Springer-Verlag, 2010.

[77] Henecka, W., Kögl, S., Sadeghi, A.-R., Schneider, T., and Wehrenberg, I., "TASTY: tool for automating secure two-party computations," in *Proceedings of the ACM conference on Computer and Communications Security*, 2010.

[78] Huang, J., *Performance and Power Characterization of Cellular Networks and Mobile Application Optimizations*. PhD thesis, University of Michigan, 2013.

[79] Huang, Y., Katz, J., and Evans, D., "Efficient secure two-party computation using symmetric cut-and-choose," in *CRYPTO*, 2013.

[80] Huang, Y., Chapman, P., and Evans, D., "Privacy-Preserving Applications on Smartphones," in *Proceedings of the USENIX Workshop on Hot Topics in Security*, 2011.

[81] Huang, Y., Evans, D., and Katz, J., "Private set intersection: Are garbled circuits better than custom protocols?," in *Proceedings of the ISOC Symposium on Network and Distributed Systems Security*, 2012.

[82] Huang, Y., Evans, D., Katz, J., and Malka, L., "Faster Secure Two-Party Computation Using Garbled Circuits," in *Proceedings of the USENIX Security Symposium*, 2011.

[83] Huang, Y., Katz, J., and Evans, D., "Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2012.

[84] Huang, Y., Katz, J., and Kolesnikov, V., "Amortizing garbled circuits," in *CRYPTO*, 2014.

[85] Hustead, N., Myers, S., shelat, a., and Grubbs, P., "GPU and CPU parallelization of honest-but-curious secure two-party computation," in *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2013.

[86] Iliev, A. and Smith, S. W., "Small, Stupid, and Scalable: Secure Computing with Faerieplay," in *The ACM Workshop on Scalable Trusted Computing*, 2010.

[87] Ishai, Y., Kilian, J., Nissim, K., and Petrank, E., "Extending oblivious transfers efficiently," in *Proceedings of the Annual International Cryptology Conference*, 2003.

[88] Ishai, Y. and Paskin, A., "Evaluating branching programs on encrypted data," in *Theory of Cryptography*, 2007.

[89] Jakobsen, T. P., Nielsen, J. B., and Orlandi, C., "A framework for outsourcing of secure computation," in *Proceedings of the ACM Workshop on Cloud Computing Security (CCSW)*, 2014.

[90] Jha, S., Kruger, L., and Shmatikov, V., "Towards practical privacy for genomic computation," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2008.

[91] Kamara, S., Mohassel, P., and Raykova, M., "Outsourcing multi-party computation." Cryptology ePrint Archive, Report 2011/272, 2011. `http://eprint.iacr.org/`.

[92] Kamara, S., Mohassel, P., and Riva, B., "Salus: A system for server-aided secure function evaluation," in *Proceedings of the ACM conference on Computer and communications security (CCS)*, 2012.

[93] Katz, J. and Malka, L., "Constant-round private function evaluation with linear complexity," in *ASIACRYPT*, 2011.

[94] Kelion, L., "Apple toughens iCloud security after celebrity breach." `http://www.bbc.com/news/technology-29237469`, 2014.

[95] Keller, M., Scholl, P., and Smart, N. P., "An architecture for practical actively secure MPC with dishonest majority," in *Proceedings of the ACM SIGSAC Conference on Computer & Communications Security*, 2013.

[96] Kerschbaum, F., "Collusion-resistant outsourcing of private set intersection," in *Proceedings of the ACM Symposium on Applied Computing*, 2012.

[97] Kerschbaum, F., Schneider, T., and Schrpfer, A., "Automatic protocol selection in secure two-party computations," in *Applied Cryptography and Network Security*, 2014.

[98] Kiraz, M. and Schoenmakers, B., "A Protocol Issue for The Malicious Case of Yao's Garbled Circuit Construction," in *Proceedings of the Symposium on Information Theory in the Benelux*, 2006.

[99] Kiraz, M. S., *Secure and Fair Two-Party Computation*. PhD thesis, Technische Universiteit Eindhoven, 2008.

[100] Kolesnikov, V. and Kumaresan, R., "Improved OT extension for transferring short secrets," in *CRYPTO*, 2013.

[101] Kolesnikov, V. and Schneider, T., "Improved garbled circuit: Free XOR gates and applications," in *Proceedings of the international colloquium on Automata, Languages and Programming, Part II*, 2008.

[102] Kolesnikov, V. and Schneider, T., "A practical universal circuit construction and secure evaluation of private functions," in *Financial Cryptography and Data Security*, 2008.

[103] Kreuter, B., shelat, a., and Shen, C., "Billion-Gate Secure Computation with Malicious Adversaries," in *Proceedings of the USENIX Security Symposium*, 2012.

[104] Kreuter, B., Mood, B., shelat, a., and Butler, K., "PCF: A portable circuit format for scalable two-party secure computation," in *Proceedings of the USENIX Security Symposium*, 2013.

[105] Kruger, L., Jha, S., Goh, E.-J., and Boneh, D., "Secure Function Evaluation with Ordered Binary Decision Diagrams," in *Proceedings of the ACM conference on Computer and communications security (CCS)*, 2006.

[106] Kushilevitz, E. and Ostrovsky, R., "Replication is not needed: single database, computationally-private information retrieval," in *Proceedings of the Annual Symposium on Foundations of Computer Science*, 1997.

[107] Larraia, E., Orsini, E., and Smart, N. P., "Dishonest majority multi-party computation for binary circuits," in *CRYPTO*, 2014.

[108] Leonard, H., "There will soon be one smartphone for every five people in the world." `http://www.businessinsider.com/15-billion-smartphones-in-the-world-22013-2`, 2013.

[109] Lindell, Y., "Fast cut-and-choose based protocols for malicious and covert adversaries," in *CRYPTO*, 2013.

[110] Lindell, Y. and Pinkas, B., "Privacy preserving data mining," in *CRYPTO*, 2000.

[111] Lindell, Y., "Lower bounds and impossibility results for concurrent self composition," *Journal of Cryptology*, vol. 21, no. 2, pp. 200–249, 2008.

[112] Lindell, Y. and Pinkas, B., "An efficient protocol for secure two-party computation in the presence of malicious adversaries," in *EUROCRYPT*, 2007.

[113] Lindell, Y. and Pinkas, B., "Secure multiparty computation for privacy-preserving data mining." Cryptology ePrint Archive, Report 2008/197, 2008. `http://eprint.iacr.org/`.

[114] Lindell, Y. and Pinkas, B., "A proof of Yao's protocol for secure two-party computation," *Journal of Cryptology*, vol. 22, no. 2, pp. 161–188, 2009.

[115] Lindell, Y. and Pinkas, B., "Secure two-party computation via cut-and-choose oblivious transfer," in *Proceedings of the conference on Theory of cryptography*, 2011.

[116] Lindell, Y. and Riva, B., "Cut-and-Choose Yao-Based Secure Computation in the Online/Offline and Batch Settings," in *CRYPTO*, 2014.

[117] López-Alt, A., Tromer, E., and Vaikuntanathan, V., "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proceedings of the Symposium on Theory Of Computing*, 2012.

[118] Malka, L., "VMCrypt: modular software architecture for scalable secure computation," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011.

[119] Malkhi, D., Nisan, N., Pinkas, B., and Sella, Y., "Fairplay–a secure two-party computation system," in *Proceedings of the USENIX Security Symposium*, 2004.

[120] Matsumoto, T., Kato, K., and Imai, H., "Speeding up secret computations with insecure auxiliary devices," in *CRYPTO*, 1988.

[121] Mersenne Research, Inc., "Great internet mersenne prime search." `http://www.mersenne.org/`, 1996.

[122] Miyaji, A. and Rahman, M. S., "Privacy-preserving data mining in presence of covert adversaries," in *Proceedings of the international conference on Advanced data mining and applications: Part I*, 2010.

[123] Mohassel, P. and Franklin, M., "Efficiency tradeoffs for malicious two-party computation," in *Proceedings of the Public Key Cryptography conference*, 2006.

[124] Mohassel, P. and Riva, B., "Garbled circuits checking garbled circuits: More efficient and secure two-party computation," in *CRYPTO*, 2013.

[125] Mohassel, P., Sadeghian, S., and Smart, N. P., "Actively secure private function evaluation," in *ASIACRYPT*, 2014.

[126] Mohassel, P. and Sadeghian, S., "How to hide circuits in MPC an efficient framework for private function evaluation," in *EUROCRYPT*, 2013.

[127] Mood, B., Gupta, D., Butler, K., and Feigenbaum, J., "Reuse it or lose it: More efficient secure computation through reuse of encrypted values," in *Proceedings of the ACM conference on Computer and communications security (CCS)*, 2014.

[128] Mood, B., Letaw, L., and Butler, K., "Memory-efficient garbled circuit generation for mobile devices," in *Proceedings of the IFCA International Conference on Financial Cryptography and Data Security (FC)*, 2012.

[129] Naor, M. and Pinkas, B., "Oblivious transfer and polynomial evaluation," in *Proceedings of the annual ACM symposium on Theory of computing*, 1999.

[130] Naor, M. and Pinkas, B., "Efficient oblivious transfer protocols," in *Proceedings of the annual ACM-SIAM symposium on Discrete algorithms*, 2001.

[131] Naor, M., Pinkas, B., and Sumner, R., "Privacy preserving auctions and mechanism design," in *Proceedings of the ACM conference on Electronic commerce*, 1999.

[132] Nielsen, J. B., Nordholt, P. S., Orlandi, C., and Burra, S. S., "A new approach to practical active-secure two-party computation," in *CRYPTO*, 2012.

[133] Nielsen, J. and Orlandi, C., "LEGO for two-party secure computation," in *Theory of Cryptography Conference*, 2009.

[134] Nipane, N., Dacosta, I., and Traynor, P., ""Mix-In-Place" anonymous networking using secure function evaluation," in *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2011.

[135] Osadchy, M., Pinkas, B., Jarrous, A., and Moskovich, B., "SCiFI-a system for secure face identification," in *Proceedings of the IEEE Symposium on Security & Privacy*, 2010.

[136] Paus, A., Sadeghi, A.-R., and Schneider, T., "Practical secure evaluation of semi-private functions," in *Applied Cryptography and Network Security*, 2009.

[137] Peikert, C., Vaikuntanathan, V., and Waters, B., "A framework for efficient and composable oblivious transfer," in *CRYPTO*, 2008.

[138] Peterson, A., "All your medical data in the cloud? Not so fast, says hhs privacy official." `http://thinkprogress.org/health/2013/01/09/1422081/medical-data-privacy/`, 2013.

[139] Pinkas, B., Schneider, T., Smart, N. P., and Williams, S., "Secure two-party computation is practical," in *ASIACRYPT*, 2009.

[140] Pinkas, B., Schneider, T., and Zohner, M., "Faster private set intersection based on OT extension," in *USENIX Security Symposium*, 2014.

[141] Pullonen, P., Bogdanov, D., and Schneider, T., "The design and implementation of a two-party protocol suite for SHAREMIND 3," Tech. Rep. t-4-17, CYBERNETICA Institute of Information Security, 2012.

[142] Qian, L., Luo, Z., Du, Y., and Guo, L., "Cloud computing : An overview," in *Proceedings of the International Conference on Cloud Computing (CloudCom)*, 2009.

[143] Rabin, M., "How to exchange secrets by oblivious transfer," Tech. Rep. TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[144] Rash, W., "Dropbox password breach highlights cloud security weaknesses." `http://www.eweek.com/c/a/Security/Dropbox-Password-Breach-Highlights-Cloud-Security-Weaknesses-266215/`, 2012.

[145] Rivest, R., Adleman, L., and Dertouzos, M., "On data banks and privacy homomorphisms," in *Proceedings of the IEEE Annual Symposium on Foundations of Computer Science*, 1978.

[146] Rivest, R., Shamir, A., and Adleman, L., "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, 1978.

[147] Ryan, M. D., "Cloud computing privacy concerns on our doorstep," *Communications of the ACM*, vol. 54, no. 1, p. 36, 2011.

[148] Sadeghi, A.-R. and Schneider, T., "Generalized universal circuits for secure evaluation of private functions with application to data classification," in *Proceedings of the International Conference on Information Security and Cryptology (ICISC)*, 2009.

[149] Schneider, T. and Zohner, M., "GMW vs. Yao? Efficient secure two-party computation with low depth circuits," in *Proceedings of the IFCA International Conference on Financial Cryptography and Data Security*, 2013.

[150] shelat, a. and Shen, C.-H., "Two-output secure computation with malicious adversaries," in *EUROCRYPT*, 2011.

[151] SHELAT, A. and SHEN, C.-H., "Fast two-party secure computation with minimal assumptions," in *Proceedings of the ACM conference on Computer and communications security (CCS)*, 2013.

[152] SHI, C., AMMAR, M., ZEGURA, E., and NAIK, M., "Computing in cirrus clouds: the challenge of intermittent connectivity," in *Proceedings of the ACM SIGCOMM Workshop on Mobile-Cloud Computing (MCC)*, 2012.

[153] SHI, C., HABAK, K., PANDURANGAN, P., AMMAR, M., NAIK, M., and ZEGURA, E., "COSMOS : Computation offloading as a service for mobile devices," in *Proceedings of The ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2014.

[154] TALBOT, D., "Security in the ether." `http://www.technologyreview.com/featuredstory/416804/security-in-the-ether/`, 2009.

[155] THOMAS, K., "Microsoft cloud data breach heralds things to come." `http://www.pcworld.com/article/214775/microsoft_cloud_data_breach_sign_of_future.html`, 2010.

[156] UNIVERSITY OF CALIFORNIA, "Seti@home." `http://setiathome.berkeley.edu/index.php`, 1999.

[157] VALIANT, L. G., "Universal circuits (preliminary report)," in *Proceedings of the Annual ACM Symposium on Theory of Computing*, 1976.

[158] WOODRUFF, D. P., "Revisiting the efficiency of malicious two-party computation," in *EUROCRYPT*, 2007.

[159] YAO, A. C., "Protocols for secure computations," in *Proceedings of the IEEE Annual Symposium on Foundations of Computer Science*, 1982.

[160] YAO, A. C., "How to generate and exchange secrets," in *Proceedings of the IEEE Annual Symposium on Foundations of Computer Science*, 1986.

VITA

Henry was born in Birmingham, Alabama to parents Fray and Susan Carter. He spent his developmental years in the town of Russellville and was educated at home with his brother Dale through high school. During this time, he earned many extracurricular honors, including Eagle Scout and a black belt in two styles of martial arts. Henry attended Belmont University for his undergraduate education, majoring in computer science and minoring in mathematics and classical piano. It was during this time that he discovered his passion for teaching, which motivated his application to the PhD program at the Georgia Institute of Technology. He plans to use his knowledge of cryptography and information security to teach future researchers and computer scientists, helping them discover and explore their own passion for science.